UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

INVESTIGAÇÃO E DESENVOLVIMENTO DE UM SISTEMA AUTOMÁTICO DE MAPEAMENTO E VISUALIZAÇÃO DE ATIVOS NUMA REDE EMPRESARIAL DE GRANDE DIMENSÃO

João Miguel Ribeiro Martins

PROJETO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Arquitectura, Sistemas e Redes de Computadores

2012

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

INVESTIGAÇÃO E DESENVOLVIMENTO DE UM SISTEMA AUTOMÁTICO DE MAPEAMENTO E VISUALIZAÇÃO DE ATIVOS NUMA REDE EMPRESARIAL DE GRANDE DIMENSÃO

projeto realizado na

Portugal Telecom

por

João Miguel Ribeiro Martins

Projecto orientado pelo Prof. Doutor Vasco Thudichum Vasconcelos
e co-orientado pelo Prof. Doutor Eng. José António dos Santos Alegria

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Arquitectura, Sistemas e Redes de Computadores

2012

Agradecimentos

Gostaria de agradecer à minha família, à minha mãe Maria Ribeiro, ao meu pai António Martins e irmãos: Dora Martins, Luís Martins, Ana Martins, Mafalda Carriço e Afonso Martins todo o apoio e conforto que me deram durante o meu percurso académico. Família que acreditou sempre em mim e que foi a minha principal fonte de apoio durante todo o meu percurso e por quem detenho um enorme carinho e orgulho.

Agradeço aos meus amigos de faculdade, que durante a licenciatura e mestrado, colaboraram comigo no estudo e na execução de dezenas de projetos. Agradeço particularmente ao Ricardo Mendes, ao Tiago Oliveira, ao Rui Nogueira, ao Diogo Simões e ao Carlos Tomás pela amizade e companheirismo que construímos durante este tempo e que foi muito importante para mim. A dedicação e espírito de grupo fantástico que empregamos em tudo o que fazíamos, apoiando-nos e motivando-nos uns aos outros, resultou num percurso académico exemplar e pleno de sucesso à custa também de muito esforço, dedicação e empenho.

Quero também agradecer a amigos, que tornaram este percurso mais fácil de suportar, à Ana Trindade, à Stephanie Martinho e ao Samuel Pereira.

Agradeço ao Prof. Doutor Vasco Vasconcelos por orientar este projeto e se mostrar sempre disponível para ajudar e contribuir para o sucesso deste, não esquecendo a orientação dada em projetos de investigação realizados e o conhecimento transmitido ao longo destes anos.

Agradeço à equipa do departamento de Eficiência, Disponibilidade e Segurança da PT Comunicações, nomeadamente ao Eng. José Alegria pela co-orientação e por ter confiado em mim para um projeto ambicioso e motivante como este, ao Eng. Pedro Inácio e ao Eng. Pedro Simões pela disponibilidade demonstrada e ao meu colega Hugo Silva que colaborou comigo neste projeto.

Por fim não quero deixar de agradecer aos profissionais que me ajudaram em detalhes técnicos e práticos deste projeto: Peter Neubauer, Mike Bostock, Andreas Ronge, Raffael Marty e Marian Steinbach.

Aos meus pais e irmãos

Resumo

Grandes organizações modernas dependem cada vez mais de grandes redes IP de comunicação para o seu funcionamento. Conhecer e controlar os dispositivos que estão efetivamente ligados nestas redes e respetivas relações e padrões de comunicação existentes é um requisito importante para a gestão da segurança.

Infelizmente, a complexidade técnica destas redes tende a aumentar dada a crescente quantidade e diversidade de dispositivos ligados e ao alto nível de mudança a que são sujeitas, muitas vezes com o aumento da dispersão geográfica.

Para uma gestão mais eficaz destas redes é importante estudar, desenhar e implementar um sistema eficiente de registo, manutenção, representação e visualização gráfica dos ativos e das suas relações de comunicação e vizinhança, recorrendo aos dados recolhidos por uma federação de sondas estrategicamente distribuídas na rede, responsáveis pela detecção de novos ativos e pela recolha de fluxos de comunicação, através de análise passiva e ativa do tráfego de rede (tema de um outro projeto de mestrado FCUL, desenvolvido paralelamente a este).

O sistema desenvolvido recorre a tecnologias e algoritmos eficientes para integrar grandes volumes de dados disponibilizados pelas diferentes sondas de rede e por outras fontes de informação que disponibilizam dados referentes a ativos importantes conhecidos, pertencentes a determinados *clusters* e subsistemas de aplicações empresariais criticas.

Estes algoritmos e tecnologias resolvem problemas de ambiguidade, incompletude e duplicação dos dados fornecidos, recorrendo a uma base de dados distribuida NoSQL e orientada a grafos que gere toda a informação através de uma modelação de modo a derivar informação mais eficazmente, facilitar a integração de novos dados e tornar as consultas (resultantes de um grande conjunto de dados e propriedades de rede genéricas) mais eficientes.

O projeto recorre a técnicas de visualização de segurança e de visualização de informação que visam tirar partido das capacidades de perceção humana através de uma interface pessoa-máquina interativa e inovadora que permite focar a atenção de uma analista de rede nos seus diferentes detalhes técnicos. A interface desenvolvida tem como principal foco a representação, visualização e interação com grafos de rede complexos referentes aos ativos, relações de comunicação e suas respetivas propriedades.

Palavras-chave: redes de computadores, monitorização, segurança, visualização de segurança

Abstract

Large modern organizations increasingly rely on large IP networks of communication for its operation. To know and to control devices that are connected to these networks and their relationships and communication patterns involved is an important requirement for network and security management.

Unfortunately, the technical complexity of these networks tends to increase given the growing number and variety of connected devices and the high level of network changes, often with increasing geographic dispersion.

For easier management of computer networks it is important to study, design and implement an efficient system of registration, maintenance, and graphic representation of the assets and their relationships and communication neighborhood. The input data is collected by a federation of probes strategically distributed in the network, responsible for the detection of new assets and the collection of communication flows through active and passive network analysis (subject of another Master's project FCUL developed).

The developed system uses efficient algorithms to integrate large datasets sent by a collection of network probes as well as other sources that provide information on known assets belonging to clusters and subsystems of important business applications.

The project uses algorithms and technologies that solve problems of ambiguity, incompleteness and duplication of data, using an NoSQL graph database which manages all the information to effectively extract new information, facilitating the integration of new datasets and efficient queries.

The project uses security visualization and information visualization techniques to exploit the capabilities of human perception through a innovative and interactive web interface providing several computer network details. The main focus of this web interface is visualization and interaction with complex network graphs, in particular assets and their communication patterns.

Keywords: network monitoring, security visualization, computer network, security

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Redes de computadores estão a tornar-se maiores e mais complexas no seio de grandes organizações empresariais. Isto deve-se aos crescentes avanços tecnológicos, à grande variedade de dispositivos interligados, ao aumento da dispersão geográfica, às expansões empresariais e à rapidez com que tais estruturas são alteradas e ajustadas às realidades de negócio.

Conhecer e controlar os dispositivos que estão efetivamente ligados nestas redes e respetivas relações e padrões de comunicação existentes é um requisito importante para a gestão da segurança.

Analistas de segurança têm necessidade de rever ficheiros de registo, rever o estado corrente de uma rede, colaborar com múltiplos grupos de trabalho, entre outros, para formar um modelo de eventos de segurança. Navegar por grandes quantidades de dados é crucial para encontrar informação e assim explorar detalhes técnicos relevantes de segurança [39]. O tempo necessário para este tipo de análise precisa de ser minimizado para fornecer uma resposta cada vez mais rápida e assim promover a mitigação dos impactos.

Uma vez que é mais fácil transmitir informação através de uma imagem do que recorrendo a uma explicação textual, é importante ter em consideração o processo de cognição humano que forma um massivo processador paralelo e que oferece uma largura de banda bastante superior para a transmissão e codificação da informação em relação aos tradicionais resultados textuais.

Algumas ferramentas de monitorização de rede e de eventos segurança, têm ao longo do tempo adicionado capacidades para reportar informação para os analistas de uma forma cada vez mais eficiente. No entanto, os resultados reportados por estas ferramentas recorrem normalmente a *logs* ou listagens textuais e raramente recorrem a abordagens visuais e interativas que facilitam a transmissão de informação.

Nos últimos anos, o campo de visualização de segurança emergiu por forma a fornecer

novas formas de mostrar informação de segurança, contribuindo deste modo para uma melhor compreensão e análise de eventos, tirando partido das grandes quantidades de informação gerados por ferramentas de monitorização e de análise de tráfego de rede.

Existem algumas ferramentas que geram representações visuais de alguns detalhes técnicos de rede. No entanto, tais visualizações estão associadas a resultados de uma ferramenta específica, mostrando apenas um subconjunto das propriedades essenciais necessárias a um analista de segurança, estando assim muito longe de integrar vários resultados produzidos por diversas ferramentas de monitorização de rede existentes e que hoje em dia se mostram essenciais a um analista de segurança. Para além disto, grande parte delas não consegue lidar com a enorme quantidade de dados gerados por redes empresariais de grande dimensão.

1.2 Objetivos

Os objetivos deste projeto consistem na investigação, desenho, desenvolvimento e avaliação de um sistema de manutenção, representação e visualização do cadastro de grandes redes empresariais, permitindo interactivamente focar a atenção de um analista em diferentes propriedades dos dispositivos de rede e das suas relações de comunicação.

Estas representações devem permitir partir da visualização global (procurando representar/abstrair um conjunto de propriedades genéricas e úteis da rede), até à visualização de todos os detalhes técnicos (das características dos dispositivos que compõem a rede e das suas relações de comunicação).

É necessário também manter o cadastro da rede com informação atualizada, integrando os resultados de uma federação de sondas (tema de outra tese em desenvolvimento) estrategicamente distribuídas na rede e responsáveis pela deteção de novos ativos e pela recolha de fluxos de comunicação. Sondas estas que geram volumes de dados enormes, sendo assim necessário recorrer a métodos eficientes para lidar com tais quantidades, resolvendo possíveis ambiguidades e duplicação de informação.

Dentro de uma rede empresarial de grande dimensão existem naturalmente certos ativos de rede mais importantes do que outros, nomeadamente ativos pertencentes a aplicações ou subsistemas críticos ao regular funcionamento da empresa. É necessário garantir um maior índice de segurança e de controlo destes dispositivos. Neste contexto é requisito desenvolver técnicas eficientes de procura de relações de vizinhança diretas entre os dispositivos críticos e os restantes ativos de rede e com isto perceber quem são estes ativos e de que forma interagem com os sistemas críticos, deixando o problema de deduzir eventuais problemas existentes para o analista.

É objetivo também a implementação de técnicas de visualização de segurança e de visualização de informação que procuram tirar partido das capacidades de perceção humana através de uma interface pessoa-máquina interativa e inovadora. A interface preten-

dida tem como objetivo disponibilizar visualizações globais e detalhadas da rede e das respetivas relações de comunicação entre dispositivos.

Para possibilitar que um analista detecte a existência de possíveis problemas de segurança, é necessário que a interface pessoa-máquina recorra a grafos de rede que apresentem diferentes níveis de abstração de informação, desde a informação mais genérica, até à mais detalhada, tirando o máximo partido dos dados que são fornecidos a este projeto.

Este é um projeto que irá integrar a plataforma Pulso de monitorização desenvolvida na Portugal Telecom.

1.3 Método de investigação utilizado

O método utilizado para a investigação e exploração de diferentes temáticas e respetivas soluções essenciais à realização do projeto foi a *Action Research*[1].

Trata-se de uma metodologia baseada num processo refletivo e progressivo de resolução de problemas. O método é cíclico e é composto por cinco fases:

Diagnosing Identificação e definição do problema;

Action Planning Exploração e criação de uma base cognitiva, considerando as diversas alternativas existentes;

Taking Action Seleção de um método de resolução e respectiva implementação;

Evaluating Avaliação e estudo das consequências de uma solução;

Specifying Learning Identificação e especificação de aprendizagens e possíveis melhorias.

Em cada fase da iteração são retiradas elações e aprendizagens, assim como hipóteses de melhoria, através de um processo de iterativo de refleção, como mostra a figura 1.1.

1.4 Enquadramento institucional

A Portugal Telecom[2] é um operador global de telecomunicações líder a nível nacional. Atua nas áreas de comunicações fixas, móveis, multimédia, sistemas de informação, investigação e desenvolvimento, comunicações via satélite e investimentos internacionais.

A PT Comunicações[3] é uma empresa do grupo Portugal Telecom, que dispõe de uma infraestrutura para proporcionar aos seus clientes mais e melhores serviços e meios de comunicação.

[1] `http://www.actionresearch.net/`
[2] `http://www.telecom.pt/`
[3] `http://www.ptcom.pt/`

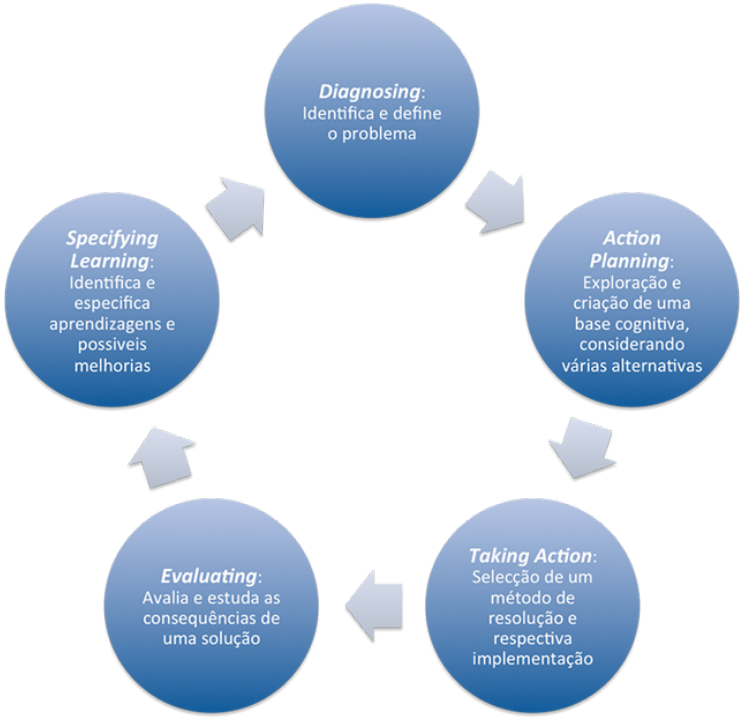

Figura 1.1: Action Research

Para aumentar os índices de produtividade e competitividade, tendo em conta a constante evolução das tecnologias de informação e a preocupação com aspetos que garantam a sua eficiência, disponibilidade e segurança, foi criado o departamento de Eficiência, Disponibilidade e Segurança dos Sistemas de Informação e/ou Tecnologias de Informação (EDS-SI/TI), departamento onde se enquadra este projeto e onde atualmente está a ser desenvolvido o sistema Pulso, descrito mais a frente na secção 2.1.

1.5 Resumo do trabalho realizado

Durante a realização deste projeto foram realizadas várias tarefas importantes, segue-se um breve sumário do trabalho realizado:

- Estudo referente ao trabalho desenvolvido anteriormente na PT Comunicações no âmbito deste projeto. Pesquisa e investigação de possíveis inovações e soluções a implementar neste projeto:
  - Investigação e estudo teórico/prático da arquitetura da federação de sondas de rede existentes e sistemas associados, nomeadamente projetos realizados anteriormente, através da leitura de algumas teses e artigos relacionados e desenvolvidos anteriormente na PT Comunicações [1, 9, 26, 32];
  - Investigação e pesquisa sobre os temas de segurança, visualização de segurança, visualização de informação e monitorização de tráfego de rede em larga

escala recorrendo a bibliografias indispensáveis e à leitura/análise de artigos e relatórios técnicos;

- Investigação exploratória relativa à integração de componentes de visualização de segurança, por forma a disponibilizar uma interface pessoa-máquina interativa e inovadora; integração de diversos algoritmos de modo a melhorar o tratamento e a gestão de dados recolhidos por uma federação de sondas de rede, com vista a facilitar a análise e motorização de uma grande variedade de casos de uso através de técnicas de interação para responder da melhor forma a uma sempre crescente quantidade de dados recolhidos em ambientes IT;
- Teste e avaliação de diversas ferramentas no âmbito da visualização de segurança, tais como o AfterGlow, GraphViz, Gephi, e bibliotecas como D3.js, Arbor.js, InfoVis e Flare de modo a avaliar as diferentes possibilidades de integração neste projeto;
- Migração de parte da plataforma anterior Pulso/Discovery, desenvolvida em Ruby on Rails v2.3, para a versão 3.1. Trabalho que requereu um estudo inicial sobre a linguagem Ruby, assim como a familiarização com modelo de desenvolvimento MVC integrado no Ruby on Rails;
- Estudo referente a bases de dados NoSQL, nomeadamente bases de dados orientadas a grafos para a modelação dos dados provenientes de ferramentas de analise ativa e passiva de tráfego de rede.

- Desenho e implementação de um *back-end* eficiente para o servidor central da nova plataforma Pulso/Discovery, referente à federação de sondas de rede:
  - Desenho e implementação de um sistema eficiente e capaz de receber grandes volumes de dados provenientes de uma federação de sondas em número arbitrário, que recorre a diferentes ferramentas para captura de tráfego de rede;
  - Construção de algoritmos eficientes para o tratamento dos dados fornecidos pelas sondas e para o respetivo armazenamento numa base de dados orientada a grafos. Algoritmos com a capacidade de resolver problemas de ambiguidade e duplicação em volumes muito grandes de dados para um posterior armazenamento que tem em consideração a interface pessoa-máquina, que necessita de navegar por grandes volumes de dados e de informação com tempos de resposta o mais curtos possíveis;
  - Estabelecimento da relação entre dados fornecidos pelas sondas Pulso/Discovery e os metadados provenientes de consultas à base de dados da plataforma Pulso (secção 2.1). Metadados que definem *clusters*, subsistemas e ativos de aplicações existentes na rede empresarial da PT Comunicações e que são importantes de monitorizar;

  - Implementação de um modelo de dados orientado a grafos, distribuído e capaz de suportar o armazenamento de grandes quantidades de informação, por forma a representar estruturas arbitrárias através de nós, relações e propriedades que facilitem posteriores consultas que retratem casos de uso necessários em diferentes níveis de abstração.

- Desenho e desenvolvimento de um *front-end web* interativo para a visualização e navegação pela informação disponível, nomeadamente:

  - Implementação de um *front-end* orientado à visualização de grafos (tal como o modelo de dados sugere) dos ativos de rede, onde os nós representam os componentes que compõem a rede e as ligações, relações de comunicação entre eles. Grafos esses que contam com a representação de propriedades relevantes, quer dos ativos, quer das comunicações, através de atributos visuais (texto, cor, forma, tamanho, níveis de transparência);
  - Implementação de filtros que permitem adicionar ou remover informação à visualização do grafo, possibilitando focar a atenção do analista em subgrafos específicos ou propriedades relevantes, gerindo assim o nível de complexidade de informação a visualizar em cada momento. Grafos estes que permitem um navegação interativa;
  - Implementação de um motor de pesquisa eficiente que permite a pesquisa por um ativo específico ou por um subconjunto de ativos de rede tais como uma subredes ou componentes pertencentes a determinados *clusters* de aplicações, assim como as respetivas relações de comunicação com origem ou destino nestes. Pesquisa por endereço IP ou gamas de endereços;
  - Implementação de uma interface para o controlo visual do grafo de rede *force-directed* (secção 3.4) em níveis físicos, controlando distâncias entre nós, níveis de repulsão e fricção, entre outros. Isto permite ajustar o posicionamento dos elementos do grafo no espaço, possibilitando ao analista deduzir informação do grafo mais eficazmente;
  - Visualização de todos os detalhes dos elementos apresentados no grafo de rede, no caso dos nós é dado por exemplo os tipos de dispositivos, sistemas operativos, marcas, endereços de IP, *hostnames*, serviços, aplicações a que pertence; no caso das ligações é dado o tipo de tráfego trocado, o volume (em número de *bytes* e pacotes), portas e protocolos utilizados e *timestamps*, entre outros.

- Teste experimental do protótipo desenvolvido e avaliação da escalabilidade, desempenho, eficiência e interação com o utilizador.

As tarefas enumeradas foram realizadas recorrendo ao método de investigação descrito na secção 1.3, pelo que muitas delas não foram ponderadas aquando do início do projeto, resultando sim de um trabalho de estudo, investigação e avaliação, que de uma forma iterativa visaram cumprir com os objetivos da plataforma Pulso/Discovery desenvolvida.

Algumas das tarefas foram planeadas e ponderadas no início do projeto, no entanto, muitas foram revistas e refinadas ao longo do desenvolvimento.

1.6 Organização do documento

Este documento segue a seguinte estrutura:

Capítulo 2 Introduz o âmbito e o enquadramento em que este projeto foi desenvolvido, apresentando o sistema Pulso atualmente em desenvolvimento pelo departamento de Eficiência, Disponibilidade e Segurança dos Sistemas de Informação e/ou Tecnologias de Informação (EDS-SI/TI) na Portugal Telecom.

Capítulo 3 Apresenta o conceito e técnicas de visualização de segurança, monitorização de dados de rede recorrendo a ficheiros de registo, representação de grafos em diagramas *force-directed* e as suas vantagens e desvantagens. Descreve bases de dados orientadas a grafos NoSQL para representação de informação como modelo de dados alternativo e apresenta também uma síntese de metodologias e tecnologias utilijadas em projetos com motivações e idênticas a este.

Capítulo 4 Introduz o projeto Pulso/Discovery anteriormente desenvolvido na PT Comunicações nomeadamente o seu foco funcional e sua arquitetura. Projeto a que se pretende dar um salto em termos funcionais por forma a responder a novos requisitos impostos pelos objetivos já referidos na secção 1.2. Aqui descreve-se em concreto quais os problemas que se pretende resolver nesta nova versão, não esquecendo o trabalho anteriormente realizado e que se mostrou útil para alguns departamentos da empresa. São identificados os aspetos relevantes em que a solução deste projeto se baseia.

Capítulo 5 Procura apoiar-se nos assuntos descritos nos capítulos anteriores para apresentar o desenho de uma solução capaz de endereçar os objetivos propostos, baseando-se no estudo efetuado e na análise do problema. Descreve também as tecnologias em que esta solução se apoia e o porquê das suas escolhas.

Capítulo 6 Introduz os três módulos desenvolvidos neste projeto, explicando como foram implementados, bem como as bibliotecas utilizadas. É apresentado a arquitetura do sistema e explicado como interagem os diferentes módulos por forma a colaborarem e fornecerem ao utilizador final a capacidade funcional a que se propôs este projeto.

Capítulo 7 Descreve os procedimentos e o ambiente em que a aplicação desenvolvida foi testada, quer em termos funcionais, quer em termos de desempenho e escalonamento. São apresentados resultados concretos do funcionamento da plataforma em ambiente empresarial, assim como os respetivos dados estatísticos referentes ao desempenho e à informação consultada da base de dados.

Capítulo 8 Conclui a tese, discutindo e esclarecendo em pormenor alguns tópicos importantes referidos nos capítulos anteriores e descreve possíveis melhoramentos e aperfeiçoamentos da plataforma num trabalho futuro. Apresenta alternativas ponderadas para a realização deste projeto, discutindo-se as razões em que se basearam as diferentes escolhas e opções. Por fim são sintetizadas as dificuldades enfrentas e as conclusões.

Capítulo 2

Enquadramento do projeto

A PT Comunicações tem vindo ao longo dos anos a interessar-se pelo desempenho e eficiência das tecnologias de informação, desenvolvendo técnicas de medição de disponibilidade dos seus processos e serviços críticos.

A plataforma Pulso foi desenvolvida para suster mecanismos de medições próximos do tempo real de modo a monitorizar e recolher indicadores de performance dos sistemas críticos aos processos de negócio. Sistemas estes que terão especial importância neste projeto, onde se irá procurar perceber quais as relações de comunicação que os ativos que compõem estes sistemas têm dentro da rede empresarial da PT Comunicações.

2.1 A plataforma Pulso

O termo Pulso refere-se simultaneamente ao programa de projetos, plataforma técnica e ao portal que o suporta e lhe fornece uma interface externa, agregando análises de qualidade de manutenção (*Quality of Maintenance*, QoM), qualidade de proteção (*Quality of Protection*, QoP) e qualidade de serviço (*Quality of Service*, QoS) [1] .

A plataforma Pulso suporta a prevenção de incidentes informáticos assim como a rápida deteção e reação a estes, na eventualidade dos mecanismos de prevenção terem sido insuficientes. Esta é uma solução 100% *open source* que tem demonstrado ser bastante eficaz e de muito baixo custo, capaz de enfrentar os elevados desafios propostos pelas organizações que nem sempre estão dispostas a alterar os seus métodos internos com vista a manter a segurança de informação. O processo de monitorização passiva (não intrusiva) do funcionamento de sistemas, protocolos de rede e bases de dados que o sistema Pulso dispõe, oferece uma solução completa para gestão do risco técnico, segurança e controlo de sistemas de informação.

O Pulso é uma plataforma desenvolvida para a deteção, recolha, análise e comunicação de eventos técnicos relevantes ao nível da qualidade, risco técnico e segurança dos principais serviços, sistemas e infraestruturas informáticas (SI/TI) de suporte ao funcionamento regular da empresa.

2.1.1 Agentes e sondas

A plataforma de monitorização Pulso, é compreendida por agentes de monitorização e sondas de rede (cliente/servidor e servidor/servidor).

Os agentes têm como função capturar periodicamente os comportamentos básicos dos seus *hosts*. Por exemplo, agentes Linux capturam variáveis tais como utilização de CPU, carregamento do CPU (para 1,5 e 15 minutos) e a utilização da memória. Para base de dados os agentes capturam tempos de processamento de requisições e, dependendo das exigências dos processos, os agentes são desenvolvidos para observar mais variáveis.

As sondas monitorizam passivamente o tráfego entre servidores e entre servidores e clientes. Estes capturam dados tendo em consideração o consumo de largura de banda e o tráfego específico de protocolos (como por exemplo requisições e erros de uma transação SQL, ou de uma comunicação IIS).

2.1.2 Objectivos

O Pulso tem como objetivo obter, de forma automática e constante, o risco técnico de todos os sistemas de informação mais importantes para o negócio e respetivas infra-estruturas tecnológicas de suporte, relativamente à sua QoS, QoP e QoM. Outro objetivo consiste na deteção e reação em tempo útil de ocorrências de incidentes relevantes ao nível de falhas técnicas, ataques externos ou internos e ao uso abusivo de recursos [32].

Ao nível da interface com a organização e com os principais responsáveis operacionais, pretende-se através de um portal colaborativo e operacional (a interface do sistema Pulso), ter um meio formal, organizado e centralizado, de disponibilização de conteúdo para a função de gestão de incidentes, segurança e controlo dos sistemas de informação.

2.1.3 Arquitetura Pulso

A plataforma serve-se da informação obtida a partir de várias fontes (computadores que disponibilizem serviços importantes, bases de dados, infraestrutura de rede, *call-center's* e lojas) para análise estatística e monitorização de diversos parâmetros. O sistema adicionalmente gera e envia alarmes às entidades que aparentam estar num estado ou situação fora do normal.

A arquitetura tem como base um subsistema distribuído para medir os parâmetros de qualidade da rede em locais considerados críticos para os processos de negócio da PT Comunicações. Tendo em conta a assimetria e a heterogeneidade da distribuição geográfica dos locais de acesso e dos sistemas numa empresa de grande dimensão como a PT foram definidos três tipos de componentes: as sondas de rede (*collectors*), os agentes de sistema (*targets*) e os *loggers* (figura 2.1).

Os agentes estão colocados em sistemas cuja qualidade de serviço se pretende monitorizar, ou seja, em locais considerados relevantes para o negócio, tendo em conta o número

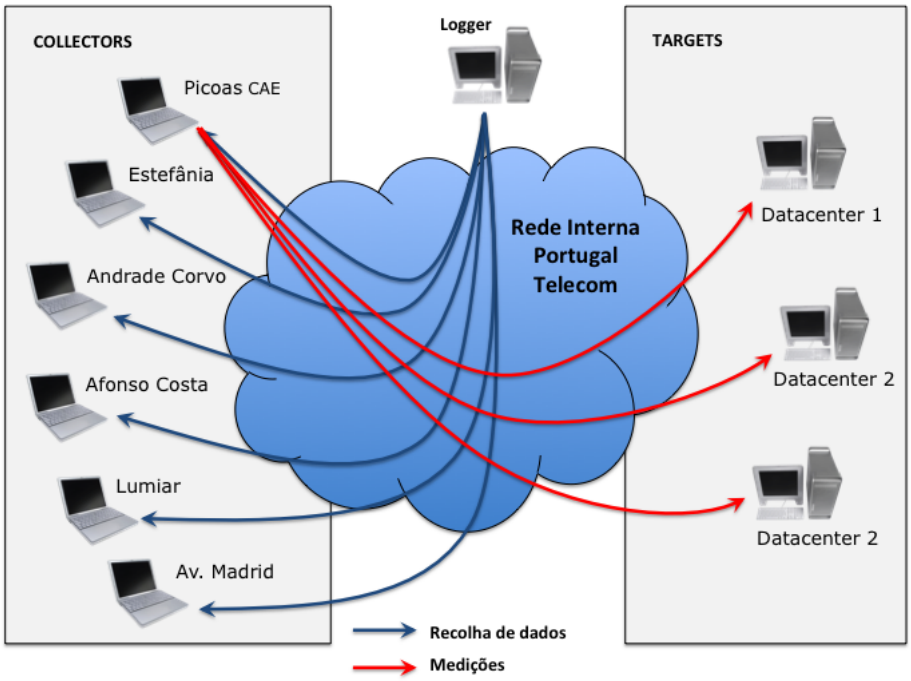

Figura 2.1: Arquitetura Pulso: composto por *collectors*, *loggers* e *targets*

de utilizadores e aplicações utilizadas.

As sondas de rede realizam medições relativamente aos agentes de sistema, monitorizando e medindo parâmetros de rede na ligação lógica compreendida entre ela e cada agente. Cada agente apenas responde aos pedidos de medição das suas sondas.

Os *loggers* têm a função de recolher periodicamente os resultados das medições que estão armazenados em máquinas chamadas *datasources* (máquina que armazena temporariamente informações do sistema Pulso). No caso da medição de rede os *datasources* correspondem às sondas de rede. O armazenamento dos resultados das medições para a base de dados do Pulso é também uma tarefa dos *loggers*.

Cada sonda está registada como *datasource* no Pulso e por isso, o módulo de recolha armazena os eventos medidos no repositório de eventos do Pulso. Existe ainda um componente que permite correlacionar a informação recolhida com acontecimentos de um número mais elevado de abstração (*Real Time Correlation Engine*) para depois ser processada em relatórios, gráficos e modelos estatísticos (*Analytical Forecasting Engine*) que compõem o portal Pulso com vários indicadores de medida para efeitos de monitorização. Por fim são ainda produzidos e enviados para a equipa técnica, alarmes a reportar todas as situações críticas ocorridas.

2.2 Configuration Management Database

Em grandes redes empresariais, a informação de todos os componentes e as suas respetivas configurações e relações são guardadas num repositório denominado CMDB (Configuration Management Database) que, para além de ter uma enorme importância técnica, revela-se uma ferramenta importante de controlo devido ao nível alto de detalhe que detém.

Um CMDB é um repositório de informação que armazena e gere os dados de todos os componentes que compõem um sistema de informação. É composto pela informação detalhada relativa a todos os componentes usados nos diferentes serviços, assim como a relações entre estes.

Este repositório disponibiliza uma visão organizada dos dados e métodos eficientes para consultar e perspetivar informação. Um componente é referido geralmente como um CI(*Configuration Item*) que pode ser um dispositivo físico, um componente de software, uma documentação, podem ser indivíduos ou qualquer combinação de um dos anteriores.

Os processos de manutenção procuram monitorizar, especificar e verificar todos estes componentes por forma a detetar alterações. Assim, é necessário que os processos de manutenção sigam práticas bem definidas, sendo quatro das principais tarefas usadas nesses processos as seguintes:

- Identificação de CI‘s para que possam ser incluídos na CMDB;
- Realização de um controlo de confidencialidade dos dados;
- Manutenção do repositório de CI‘s atualizado;
- Realização de auditorias e revisões aos dados do repositório, por forma a garantir exatidão.

Com o rápido crescimento e evolução a que uma rede empresarial está sujeita, a manutenção destes repositórios torna-se um problema devido à necessidade constante de atualização e verificação, sobretudo quando todo esse trabalho tem de ser feito manualmente.

É também por estas razões que uma plataforma como o Pulso/Discovery (plataforma à qual este projeto pretende dar um salto funcional) surge, mostrando-se um forte aliado à CMDB, devido à automatização dos processos de descoberta e monitorização graças à fiabilidade que as técnicas de análise de tráfego possuem.

Embora a plataforma Pulso/Discovery esteja longe de fornecer uma informação completa sobre um determinado CI, as suas funcionalidades poderão tornar-se úteis para processos de manutenção da CMDB. Inversamente, a plataforma pode também futuramente tirar partido da base de conhecimento da CMDB.

2.3 Aplicações SOX e aplicações Pulso

As aplicações SOX (nome derivado da lei denominada Sarbanes-Oxley) visam a criação de mecanismos de auditoria e segurança confiáveis na PT Comunicações, evitando a ocorrência de fraudes ou assegurando que existem meios para identifica-las caso ocorram, garantindo a transparência na gestão da empresa. Aplicações que operam em *clusters* bem definidos, havendo necessidade de monitorizar a sua atividade de rede de modo a haver um maior controlo de segurança.

Como descrito anteriormente na secção 1.2, é requisito deste projeto tomar particular atenção aos elementos dos *clusters* destas aplicações, para haver um maior controlo e conhecimento dos ativos (internos, externos ou mesmo desconhecidos) que interagem com estes *clusters*. Com base nos dados provenientes da análise de tráfego de rede onde as estes *clusters* se encontram, é necessário produzir informação útil, possibilitando conhecer eventuais problemas de segurança e com isso, promover procedimentos para mitigar ou eliminar esses problemas.

Além das aplicações SOX, existem outras aplicações que disponibilizam serviços importantes aos processos de negócio da PT comunicações e que o sistema Pulso monitoriza, quer para análise estatística, quer para controlo da disponibilidade e segurança das mesmas. Aplicações e subsistemas esses para os quais a plataforma Pulso/Discovery (desenvolvida neste projeto) também terá um papel relevante pois terá a responsábilidade, tal como acontece com as aplicações SOX, de analisar relações de comunicação dos dispositivos que compõem estes subsistemas, permitindo perceber por exemplo, se um dado fluxo de comunicação é expectável de existir ou não numa dada aplicação, *cluster* ou subsistema.

Os metadados de aplicações, subsistemas e *clusters* monitorizadas pelo Pulso são usados neste projeto com o propósito de dar um maior foco aos ativos que os compõem e distingui-los dos restantes descobertos por análise passiva e ativa de rede.

Capítulo 3

Trabalho relacionado

Tendo este projeto o objetivo de inovar a plataforma Pulso/Discovery existente, refazendo funcionalidades e melhorando o modo de navegação pela informação através de visualizações úteis que realcem propriedades de rede importantes através dos seus componentes e das suas relações de vizinhança, é necessário fazer uma introdução aos conceitos e técnicas estudadas que tiveram particular importância no desenvolvimento da resolução proposta.

3.1 Visualização e os seus objetivos

Segundo Colin Ware, devemo-nos interessar pela visualização pois o sistema visual humano é poderosíssimo e subtil na procura e identificação de padrões [39]. O olho e o córtex visual do cérebro, formam um massivo processador paralelo que oferece uma "largura" de banda enorme para os centros de cognição humano. As imagens podem usar forma, cor, tamanho, posicionamento relativo, entre outros para a codificação da informação, contribuindo para o aumento da "largura de banda" entre a informação e o consumidor ou observador, "largura de banda" esta que a nova versão Pulso/Discovery terá que levar em linha de conta pois a informação que é necessária transmitir é imensa mesmo quando se recorre a abstrações e agregações.

Muitas áreas enfrentam problemas de *ever-growing* de dados, que precisam de ser analisados, processados e comunicados. Uma grande percentagem de informação é guardada em forma de texto, base de dados, documentos, *websites*, emails, entre outros. São necessárias novas formas de trabalhar com toda esta informação. Os analistas que precisam de observar, navegar ou perceber a informação, têm a necessidade de representar a informação relevante graficamente, assistindo assim à compreensão, à análise ou simplesmente para memorizar partes.

Navegar por grandes quantidades de dados é crucial para encontrar informação e assim explorar detalhes [39]. A interação é um dos elementos chave neste processo. Não é apenas a capacidade de navegação acelerada que a visualização tem para oferecer, mas ge-

ralmente uma representação visual, em contraste com a representação textual, ajuda a descobrir relações escondidas no volume de dados. Um exemplo simples de uma aplicação é o LinkedIn Maps[1], uma aplicação LinkedIn que gera visualizações de relações e ligações entre utilizadores.

A visualização tornou-se importante nos sistemas de análise de segurança, oferecendo vantagens tais como [39]:

- Fazer emergir propriedades que não são antecipadas ou previstas;
- Tornar aparente os problemas com os dados;
- Auxiliar à perceção de características de larga escala ou pequena escala;
- Facilitar a formulação de hipóteses.

Os dados de eventos de segurança são massivos e abrangem vários dispositivos que requerem colaboração entre os seus administradores e entre si. A visualização pode fornecer significado e melhorar o processo de correlação.

Embora haja necessidade de visualização de informação em diferentes disciplinas, a explosão de informação existente afeta a segurança informática mais do que em outras áreas. A análise de segurança enfrenta um crescimento constante de dados que precisam de ser analisados e dominados. Hoje em dia existe a necessidade de analisar várias camadas do modelo OSI, começando na camada de rede e seguindo até ao topo às aplicações, que são particularmente boas a gerar volumes de dados intratáveis.

3.2 Visualização de segurança

Ficheiros de *log*, ficheiros de configuração e outras informações de segurança devem ser analisados e monitorizadas para responder a uma grande variedade de casos de uso. A visualização, em contraste com a gestão direta da informação em forma de texto como acontecia com a versão anterior Pulso/Discovery, oferece uma abordagem nova, mais eficiente e mais simples para analisar milhões de registos de entrada gerados diariamente.

Representações gráficas ajudam a identificar mais eficazmente discrepâncias, detetar atividades maliciosas, desmascarar anomalias e más configurações e realçar tendências e relacionar diferentes pontos de informação. Como referido por Daisuke Mashima, Stephen G. Kobourov e Yifan Hu a frase "*I saw it with my own eyes*" tem de ser usada pelos peritos que analisam este tipo de informação, é necessário que estes olhem para uma imagem e tenham essa experiência [27]. Uma imagem diz mais do que mil palavras, isto é, uma representação visual da informação pode comunicar muitos detalhes de uma forma que é acessível instantaneamente e com significado.

[1] `http://inmaps.linkedinlabs.com/`

Não é só um maior volume de informação que pode ser transmitido, há um fenómeno muito mais interessante chamado *critical faculty* ou o *skepticism filter* [4]. Ao olharmos para uma imagem, este filtro de ceticismo não existe num primeiro momento, no entanto, quanto mais profundamente observarmos, mais detalhes começamos a ver, e mais convencidos ficamos.

Segundo Raffaell Marty, as imagens são usadas eficientemente para comunicar informação. "*A picture is worth a thousand log records*" [27]. Ao invés de o analista ter que interagir diretamente com ficheiros de registos/*logs* ou informação textual que descreve por exemplo, como um ataque aconteceu, podemos usar uma imagem ou uma representação visual. De uma forma rápida, uma imagem pode comunicar o conteúdo de um ficheiro de log. Os utilizadores ou analistas podem assim processar a informação numa fração de tempo reduzida, quando comparado à leitura do ficheiro de registo original. A visualização, num sentido de segurança, é o processo de gerar uma imagem baseado em dados de ficheiros de registo. Este define como os registos de log são mapeados numa visualização gráfica.

O campo de visualização de segurança é muito recente. Até à data, foi feito muito pouco trabalho nesta área. Dado a enorme quantidade de dados necessários para analisar problemas de segurança, a visualização parece ser a abordagem mais correta. A sempre crescente quantidade de dados recolhidos em ambientes de tecnologias de informação, está sujeito à procura de novos métodos e ferramentas para lidar com estes e os tornar efectivamente úteis. A análise de eventos e *logs* está a tornar-se umas das principais formas de análise de segurança para investigar e compreender o estado de uma rede, como *hosts*, aplicações e processos de negócio. Todas estas tarefas lidam com quantidades de dados enormes que precisam de ser analisados.

Devido à vasta quantidade de dados existentes com necessidade de análise, algumas ferramentas de segurança, tais como *firewalls* e sistemas de deteção de intrusões (IDS), têm ao longo do tempo adicionado capacidades para reportar informação para os utilizadores. No entanto, as ferramentas atuais para administradores de sistemas e analistas de segurança oferecem geralmente informação na forma de texto, tipicamente em *logs* ou em forma de tabelas o que dificulta a análise. Foi por estes motivos que nos últimos anos, a área de visualização de segurança emergiu com o objetivo de fornecer novos métodos de apresentação da informação de segurança por forma a facilitar a sua compreensão e análise.

Atualmente os produtos de segurança ainda não são desenhados com a visualização em mente, no entanto, esta situação está a melhorar embora que muito lentamente. O problema que estas ferramentas apresentam é que são fortemente especializadas, lidando apenas com dados gerados por uma ferramenta específica e geralmente, é necessário visualizar dados provenientes de múltiplas ferramentas que se complementam em termos de informação necessária a um analista de segurança.

Motivado pela crescente necessidade de métodos e ferramentas automáticas de visualização, a visualização de segurança emergiu como uma comunidade interdisciplinar de investigadores denominada VizSec[1].

Muitas empresas (entre elas a PT Comunicações) estão a aperceber-se que a visualização é uma vantagem competitiva e que as suas tarefas são significativamente simplificadas com as ajudas visuais. São precisos novos métodos para gerir dados provenientes de grandes listagens textuais como *logs* e analisar dados de segurança.

3.2.1 Benefícios

A análise de grandes ficheiros com dezenas de milhares de entradas é difícil. Uma abordagem visual facilita a tarefa significativamente. A visualização oferece benefícios em relação à análise textual. Estes benefícios são baseados na habilidade das pessoas em processar as imagens eficientemente. É possível procurar, reconhecer padrões e mais eficazmente memorizar imagens. Segue-se um breve sumário dos benefícios das técnicas de visualização de segurança:

Responde a questões A visualização possibilita criar uma imagem para cada questão que se proponha ou para cada informação que se pretenda obter a partir de um conjunto de dados. Ao invés de percorrer os dados textualmente e tentar relembrar todas as relações entre registos pode-se usar uma imagem que condensa os dados de uma forma concisa.

Coloca novas questões Um aspeto interessante nas representações visuais é que possibilitam ao utilizador que as vê, gerar novas questões. O analista, ao observar, tem a capacidade de identificar padrões. Geralmente, esses padrões não são antecipados no momento que a visualização é gerada. Questões como: "Que discrepância é esta?" "Qual a razão destas máquinas comunicarem entre si?".

Explora e descobre Uma representação visual fornece novas perceções dentro de um dado conjunto de dados. Diferentes representações e configurações de realce de determinadas propriedades ajudam a identificar informação não derivada anteriormente. Se as propriedades e relações forem conhecidas de antemão, será possível identificar incidentes sem recorrer à visualização. No entanto, estes têm de ser descobertos primeiro, e ferramentas de visualização ajudam neste processo. Visualizações interativas ajudam a descobrir propriedades escondidas num determinado conjunto de dados.

Suporte à decisão A visualização ajuda a analisar um grande conjunto de dados muito rapidamente. Decisões podem ser baseadas em grandes quantidades de dados pois a visualização ajuda a refiná-los em algo com significado ou em informação útil.

[1] `http://www.vizsec.org/`

Comunicação de informação Representações gráficas de dados são mais eficientes como meio de comunicação do que as tradicionais listagens/ficheiros de *logs*. Uma sequência de eventos pode ser mais eficazmente transmitida recorrendo a uma visualização, e com tempos bastante inferiores quando comparado com a consulta de dados textuais.

Aumento da eficiência Ao invés de percorrer as milhares de linhas de texto, é muito mais eficiente representar gráficamente certas propriedades por forma a descobrir tendências e descrepâncias. Isto liberta tempo que permite aos analistas refletirem sobre padrões e relações encontradas nos dados. Acelera também a deteção e resposta a novos desenvolvimentos.

3.3 Monitorização recorrendo a ficheiros de registo

Analisar e monitorizar ficheiros de *logs* que retratam a atividade de rede e o estado dos sistemas é essencial para conhecer os requisitos de segurança e a disponibilidade de recursos [17]. Ao passo que a maioria dos sistemas possui hoje em dia boas facilidades de *logging*, as ferramentas para monitorizar e interpretar tais eventos estão ainda pouco exploradas.

Os administradores usam geralmente poucas ou nenhumas ferramentas para suportar e interpretar tais ficheiros de registo. No entanto, com o constante crescimento de sistemas interligados, variedade de dispositivos e da internet, resultou num drástico aumento da quantidade de dados, fazendo com que os *scans* sequênciais se tornassem impraticáveis. Existem algumas ferramentas baseadas em regras de sistema que eliminam registos conhecidos como irrelevantes e criam abstrações estatísticas para simplificar a análise. Tais ferramentas aliviam o trabalho repetitivo de exploração dos ficheiros de *log* mas geram outros problemas: A criação de regras força os administradores a impor uma distinção clara entre o que é relevante/irrelevante, critico/isolado, severo/benigno.

Traduzir o conhecimento tácito de um especialista num conjunto de regras formais é difícil e propício a erros, além disso, as regras resultantes apenas cobrem comportamentos conhecidos, sendo que os eventos mais interessantes são geralmente resultado de eventos imprevistos. Assim, é preferivel não recorrer a estes métodos, optando antes pelas técnicas de visualização de segurança que procuram basear-se em conjuntos de dados completos para a transmissão da informação.

3.4 Grafos de rede

Visualizar relações entre entidades que compõem os dados de segurança é um dos casos onde os grafos se tornam bastante úteis.

A visualização de relações de comunicação entre ativos de rede é um dos exemplos que permite tirar partido de representações de grafos por forma a perceber quem interage com quem. Grafos que podem ser construídos a partir dos padrões deduzidos dos ficheiros de registo/*logs* produzidos por algumas ferramentas.

Estes padrões de relacionamento entre entidades, podem ser complementados com diferentes atributos visuais que dão dimensões aos dados representados (adicionando cores, formas, transparências, e diferentes tamanhos aos elementos) traduzindo um conjunto arbitrário de propriedades destes.

Por exemplo, colorir endereços de IP de uma dada rede de computadores, recorrendo a uma cor para os endereços internos (endereços conhecidos) e a outra para os restantes (endereços externos/desconhecidos), ou usar diferentes formas/símbolos para representar diferentes tipos de ativos (impressoras, switches, computadores, servidores, etc.) podem ser formas de enriquecer a visualização dos grafos, tornando-os mais úteis a quem os analisa.

A visualização de grafos, aliada a boas técnicas de represetanção visual, ajuda a identificar imediatamente relações importantes e propriedades chave dos seus elementos, algo que numa represetação textual é mais difícil evidenciar.

3.5 Representação visual de grafos

A classe de algoritmos mais usada para a o desenho de grafos de nós ligados é a classe *force-directed* [18].

O propósito dos métodos *force-directed* consistem em posicionar os nós do grafo no espaço, de modo a que todas as arestas fiquem mais ou menos de comprimento igual e a que exista o menor número possível de arestas cruzadas. Isto é conseguido ao atribuir forças como que se as arestas fossem molas e os nós fossem partículas de energia.

O grafo inteiro é simulado como um sistema físico. As vantagens na utilização destes algoritmos para o desenho de grafos são as seguintes:

Qualidade dos resultados Em grafos de tamanho médio, os resultados são bastante bons baseados nos seguintes critérios: comprimentos de arestas uniformes, comprimentos de nós uniformes e simetria de representação. Este ultimo critério é muito importante e difícil de obter com outros tipos de algoritmos.

Flexibilidade Os algoritmos podem ser facilmente adaptados e estendidos para se adaptarem a critérios estéticos adicionais. Isto faz dele a classe mais versátil dos algoritmos de desenho de grafos. Exemplos de extensões são desenho de grafos 3D, desenho de grafos de agrupamento (de nós e arestas) e desenho de grafos dinâmicos.

Intuição Uma vez que são baseados em analogias físicas tais como molas, o comportamento dos algoritmos é relativamente simples de prever e de perceber.

Simplicidade Tipicamente algoritmos *force-directed* são simples e podem ser implementados em poucas linhas de código. Outras classes de desenho de algoritmos, como o *orthogonal layouts*, são geralmente muito mais complexos.

Interatividade Através de algumas ferramentas de desenho de grafos, o utilizador pode puxar um ou mais nós para uma diferente posição no espaço e criar um novo estado de equilíbrio físico, alterando a configuração do grafo. Isto faz com que esta classe de algoritmos seja a escolha preferida para sistemas de desenho de grafos dinâmicos na internet.

3.5.1 Oclusão em diagramas de nós e ligações

Os grafos são geralmente visualizados em diagramas de nós e ligações. Apesar deste tipo de representações de grafos fornecer uma forma intuitiva de os representar, a oclusão visual torna-se um problema à medida que estes se tornam mais complexos e procuram representar um grande número de nós e ligações num espaço limitado [18]. Isto pode ser remediado recorrendo a outro tipo de visualizações para os grafos de grandes dimensões. Por exemplo, uma representação em matriz pode ser usada como alternativa [38].

Embora representações em matriz ofereçam uma disposição limpa e sem oclusão, estas mostram-se muito menos intuitivas do que os diagramas de nós e ligações [15]. É preferível assim manter os grafos e tentar reduzir a oclusão visual que é geralmente associada a este tipo de representações em larga escala.

Várias abordagens foram tomadas para resolver este problema, uma delas procura agregar arestas através de um método que usa uma auto-organização destas, onde são moduladas como molas flexíveis que se atraem mutuamente [18]. Embora o resultado mostre uma significante redução de oclusão nas relações entre nós, esta abordagem torna-se bastante pesada em termos de complexidade computacional.

Existem outras técnicas de desenho que procuram resolver este tipo de problemas. Técnicas de *fisheye* sobre os grafos, de agrupamento dos nós/arestas com propriedades semelhantes ou de construção do grafos em diferentes níveis de profundidade, são opções válidas consoante o tipo de dados que se procura representar.

3.6 Base de dados NoSQL orientada a grafos

Um modelo de dados orientado a grafos representa a mais genérica estrutura de dados, sendo capaz de representar elegantemente qualquer tipo de dados que podem ser acedidos de uma forma bastante eficaz. Nestes modelos de dados, o grafo mais simples possível contem apenas um nó que pode registar atributos referidos como propriedades. Um nó pode começar com uma única propriedade e crescer até uns milhões de propriedades

(embora se torne um pouco confuso). A certo ponto, faz sentido distribuir os dados em múltiplos nós e organiza-los através de relações [36].

As relações organizam os nós em estruturas arbitrárias, permitindo que um grafo se assemelhe a uma lista, uma árvore, um mapa ou uma entidade composta combinando estruturas e formando uma mais complexa, enriquecendo estruturas (figura 3.1).

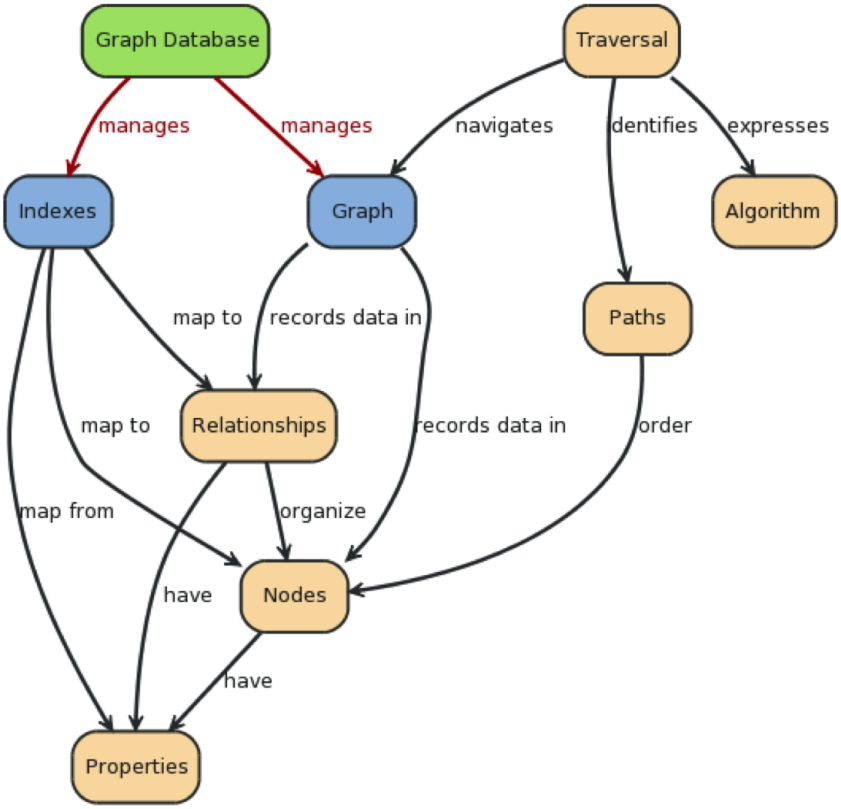

Figura 3.1: Esquema de uma base de dados orientada a grafos [36].

Uma visita, é a forma de realizar consultas a um grafo, navegando dos nós iniciais, de acordo com um algoritmo específico. Geralmente, pretende-se encontrar um nó ou uma relação específica de acordo com a propriedade que este tem. Este caso especial de visita é tão comum que é otimizado através de uma indexação.

Em comparação com base de dados relacionais, as bases de dados orientadas a grafos tendem a ser mais eficientes na presença de um alto nível de complexidade, ou seja, em estruturas de dados com baixo acoplamento, que mudam frequentemente e que sofrem frequentes consultas. Em modelos relacionais, a realização de consultas mais complexas podem gerar um grande número de uniões (*Joins*), o que se refletir em problemas de desempenho.

Durante os últimos três anos a oferta em TI mudou muito no que diz respeito a bases de dados. Muitas aplicações oferecem agora SaS (*software as a service*), e normalmente, suportado por algum tipo de *cloud computing*.

Uma base de dados relacional convencional, onde a estrutura de tabelas, colunas ou campos é predefinida, não oferece flexibilidade suficiente. As bases de dados baseadas em grafos são geralmente mais livres de estruturas e permitem que um conjunto de nós com propriedades dinâmicas (o correspondente a colunas ou atributos) sejam arbitrariamente ligados a outros nós.

Bases de dados como o Neo4j[1] e o InfoGrid[2] são opções quando á se procura flexibilidade no modelo de dados.

3.7 Metodologias de desenho utilizadas em visualização de segurança

Existem várias abordagens de desenho de visualizações, dependendo dos vários conceitos de segurança existentes. Entre os cinco mais importantes referidos por Thomas M. J. Fruchterman encontra-se a monitorização de rede.

A monitorização da atividade de rede procura identificar comportamentos tais como os altos volumes de tráfego para ou a partir de certos *hosts*, ajuda a identificar alguns tipos de ataques como tentativas de intrusão, *scans*, *worms*, ou ataques *denial of service* assim como perceber relações de vizinhança existentes entre os seus elementos [35].

Graças à sua versatilidade, as técnicas de desenho de grafos são umas das principais abordagens empregues em visualização de segurança.

3.7.1 Monitorização de rede

Foram estudadas diferentes abordagens de visualização de dados provenientes da monitorização de rede dentro da visualização de segurança.

Tolle e Niggemann usam uma combinação de desenho *force-directed* e técnicas de agrupamento num sistema de deteção de intrusões [37]. A rede de dispositivos é modelada como um grafo onde os nós são computadores e as arestas são comunicações com a largura proporcional ao tráfego de rede nessa ligação.

Os agrupamentos do grafo são realizados através de um método iterativo simples. Inicialmente cada nó forma o seu próprio agrupamento. De seguida, os nós juntam-se a agrupamentos que já têm a maioria dos seus vizinhos. Uma abordagem *force-directed* é usada para colocar agrupamentos e nós dentro de agrupamentos. Uma vez que as forças são proporcionais às larguras das arestas, se há muita comunicação entre dois *hosts*, os seus nós são colocados próximos.

Além disto, no grafo de agrupamentos, existe uma ligação entre A e B se existir pelo menos uma ligação entre algum nó do agrupamento A e algum nó do agrupamento B. A

[1] `http://neo4j.org/`
[2] `http://infogrid.org/trac/`

disposição dos elementos do grafo de agrupamentos e de cada agrupamento específico é computado usando o método *force-directed spring embedder* [11].

Mansmann, Meier e Keim, mostram como visualizar a evolução ao longo do tempo do volume e tipo de trafego de rede usando técnicas de desenho de grafos *force-directed* (figura 3.2) [24].

Uma vez que há diferentes tipos de protocolos de rede (HTTP, FTP, SMTP, SSH, entre outros) e múltiplos tipos de períodos de tempo, estes dados multidimensionais são modelados por um grafo com nós de dois tipos: nós de dimensão (denominados *dimension nodes*), que representam protocolos de comunicação e, nós de observação (denominados *observation nodes*), que representam o estado de um certo *host* num dado intervalo de tempo. As ligações são também de dois tipos: arestas de registo (denominadas *trace edges*), que ligam nós de observação de intervalos de tempo consecutivos e, arestas de atração (denominadas *attraction edges*), que ligam nós de observação com nós de dimensão e têm a largura proporcional ao tráfego do protocolo que representam.

A disposição dos elementos do grafo, é computado através de um posicionamento fixo dos nós de dimensão e de seguida, é executado uma versão modificada do algoritmo *Fruchterman-Reingold force-directed* que visa alcançar comprimentos de arestas uniformes [14]. É demostrado como é que os alertas de deteção de intrusões podem ser associados com padrões no grafo.

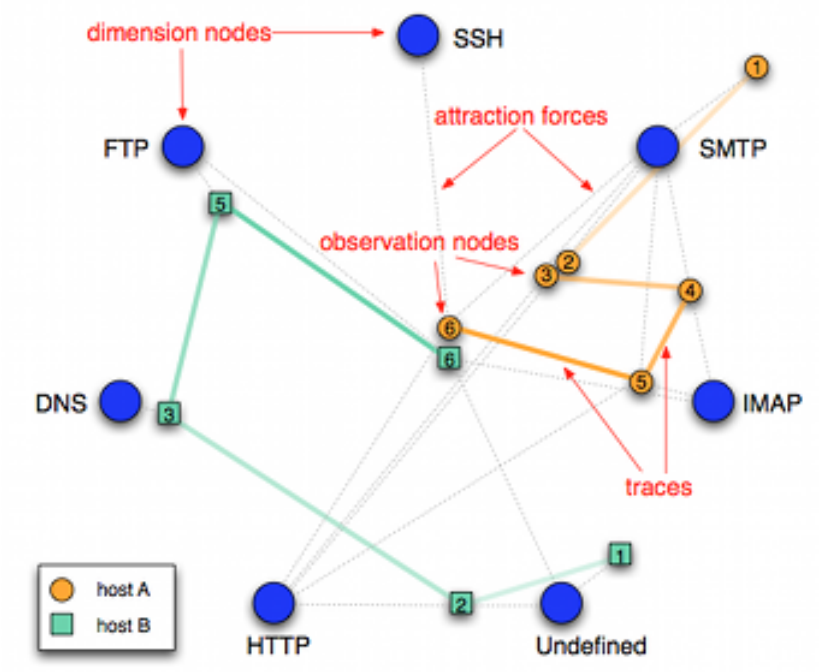

Figura 3.2: Evolução do tráfego de rede ao longo do tempo [24].

Ginardin e Brodbeck apresentam uma técnica para visualizar entradas de *logs* [17]. Estes *logs* são obtidos por monitorização do tráfego de rede (figura 3.3).

As entradas dos *logs* são basicamente vetores cujos elementos correspondem a características do trafego de rede, incluindo IP de origem, IP de destino, e volume de tr(á)fego. Os autores constrõem grafos de semelhança ponderados para as entradas dos *logs* usando uma métrica simples de distância para cada duas entradas que é dado pela soma das diferenças entre os respetivos elementos. O algoritmo *force-directed* usado é o descrito por Chamers e é utilizado para computar os grafos de similaridade [6] .

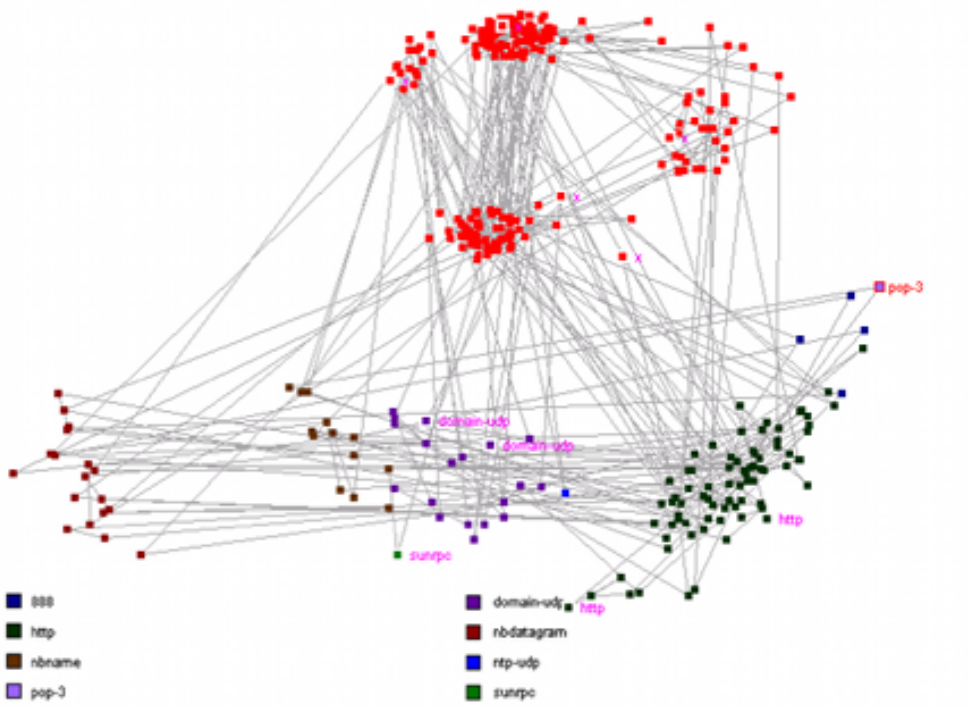

Figura 3.3: Grafos de similaridade de registos dos ficheiros de *logs* [17].

Muelder, Ma e Bartoletti focam-se em *scans* de rede, geralmente usados numa fase preliminar de um ataque [28]. Os autores desenvolveram uma sistema de visualização que mostra as relações entre diferentes *scans* (figura 3.4). É criado um grafo onde cada nó representa um scan e as ligações entre estes são ponderadas de acordo com algumas métricas (medidas de similaridade) que são definidas para os dois *scans*. As características tidas em consideração para a definição da medida de similaridade incluem o IP de origem e de destino e o tempo da conexão.

Para evitar mostrar grafos incompletos, é definido um *threshold* mínimo para as métricas medidas, as ligações são removidas quando os seus valores estão abaixo deste limite. Neste caso é usado o método *LinLog force-directed layout* para a visualização [30]. Na produção da visualização, conjuntos de *scans* similares são agrupados, facilitando assim a identificação visual de *scans* maliciosos.

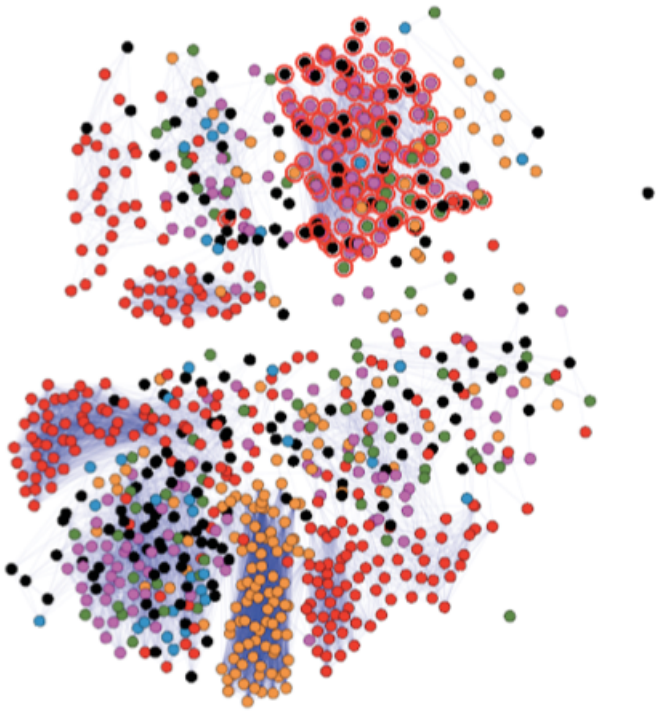

Figura 3.4: Grafos de similaridade de *scans* de rede [28].

3.7.2 Síntese

Representações visuais de redes de computadores estão disponíveis mas geralmente não escalam eficientemente para implementações de redes de larga escala. As limitações destes sistemas resultam numa sobrecarga de informação para o analista de segurança.

Sistemas de deteção de intrusões tais como o Snort e soluções comerciais como o Cisco IDS *Suite of Products*[1], oferecem alertas IDS num formato de lista [31]. Estas listas podem ser extremamente longas e difíceis de analisar. A simples ordenação da informação de colunas é insuficiente.

A representação de redes de larga escala abandona geralmente a tentativa de visualizar a rede em si e procura visualizar o tráfego [25]. Os grafos *force-directed* em conjunto com as *treemaps* surgem como um dos métodos mais eficientes para apresentar informação, no caso das *treemaps* ao dividir a representação total em subunidades para identificar a origem do trafego, o destino e o seu tipo.

Apesar de qualquer um destes sistemas apresentados anteriormente na seção 3.7 poder mostrar algum tipo de visualização de dados de rede de baixo nível de modo a reduzir a sobrecarga de informação para o analista é sempre necessário algum tipo de pré processamento de dados para os poder construir. Motores de correlação, tais como o usado numa tese anterior no âmbito do projeto Discovery, são uma das possíveis abordagens [26].

[1] `http://www.cisco.com/`

Os sistemas apresentados mostram-se também demasiado específicos em termos de resultados produzidos, procurando focar-se em problemas concretos de segurança ao invés de reproduzir visualizações que procurem representar ativos de rede descobertos e as respetivas relações de comunicação, deixando para mais tarde o problema de derivar eventuais problemas.

Capítulo 4

Análise do problema

No desenvolvimento de um projeto é importante realizar uma análise antes da conceção de qualquer desenho aplicacional. Para isso é necessário olhar em detalhe para os objetivos (capítulo 1) e para o trabalho relacionado (capítulo 3), no contexto onde este projeto se insere, e compreender os problemas que se pretende resolver para depois então, conceber um desenho capaz de ser implementado de acordo com os requisitos funcionais impostos.

É importante não esquecer todo um conjunto de funcionalidades que a ferramenta Pulso/Discovery anteriormente desenvolvida disponibilizava e perceber que problemas pretendia resolver, qual a eficiência com que o fazia e a quem se destinava. Com este conhecimento, é necessário inovar a plataforma de acordo com os objetivos a que o novo Pulso/Discovery se propõe.

Ao analisar os desafios propostos para este projeto torna-se essencial estudar alternativas capazes de responder a um conjunto de requisitos funcionais.

4.1 Versões Pulso/Discovery anteriores

Para o desenho de uma solução para este projeto é necessário analisar a ferramenta Pulso/Discovery existente e compreender a sua funcionalidade base e conhecer a sua utilidade atualmente.

A figura 4.1 mostra a arquitetura da plataforma na sua última versão antes do início deste projeto, descrevendo as interações entre as três componentes, as sondas, o servidor central e o portal *web* [26] . Como pode ser visto na figura, os dados são capturados nas sondas e enviados da aplicação Discovery Probe para a aplicação Discovery Collector, responsável por receber e armazenar os dados na base de dados, validando-os e mantendo alguma consistência. O portal Pulso/Discovery acede a esta base de dados para disponibilizar a informação para o utilizador.

A ferramenta Pulso/Discovery, no que diz respeito à parte servidor, utilizava uma base de dados relacional e uma interface pessoa-máquina baseada em JQuery, com funciona-

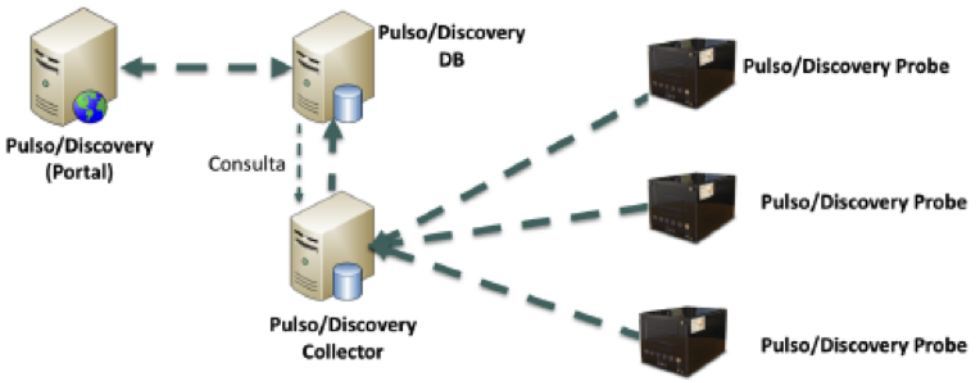

Figura 4.1: Arquitetura da plataforma Pulso/Discovery anterior [26].

lidades básicas de procura de um determinado *host*/componente e o acesso à informação mais detalhada sobre este, nomeadamente o tipo de dispositivo, sistema operativo, os respectivos fluxos de comunicação e as suas características [26].

Um projeto que se mostrou útil para a PT comunicações, nomeadamente para perceber que características tem um determinado *host* ou saber quais as comunicações diretas em que este participa. Permite por exemplo, conhecer que tipo de tráfego existe entre determinados *hosts* e com essa informação, perceber que regras aplicar a uma determinada *firewall* de modo a não interferir com o normal funcionamento dos sistemas, entre outros propósitos similares.

No entanto, a ferramenta não procurava mostrar quaisquer relações de vizinhança entre ativos de rede, ou obter uma visualização genérica de um subconjunto destes por forma a relacionar ativos.

Tornou-se importante observar a rede como um todo, realçando propriedades de rede úteis a um analista, para que este possa perceber onde pode haver eventos suscetíveis a uma atenção especial e aí então consultar informação mais detalhada, permitindo deste modo conhecer/deduzir um conjunto de problemas de segurança que possam existir. A interação com a informação do sistema Discovery anterior é limitada, não servindo os objectivos futuros que a empresa procura para a ferramenta.

A última versão do projeto Discovery, não dispõem de alguns casos de uso importantes que foram ponderados aquando na definição deste projeto, por forma a tirar partido de toda riqueza dos dados capturados por análise passiva e ativa de tráfego.

4.2 Arquitetura

Um dos aspetos necessários de analisar é a arquitetura base Pulso/Discovery para este projeto. Trata-se de uma plataforma composta por dois módulos essenciais que irão ser desenvolvidos de uma forma autónoma mas que no entanto irão ter objetivos comuns, necessitando assim de estar sincronizados.

Por um lado existe os componentes de captura ativa e passiva do tráfego de rede por parte das sondas, e por outro, os componentes que dizem respeito a este projeto, ou seja, a parte servidor, o tratamento dos dados recebidos e a respetiva interface pessoa-máquina que irá disponibilizar a informação.

A arquitetura base que servirá a plataforma Pulso/Discovery é assim dividida em dois módulos como apresenta a figura 4.2.

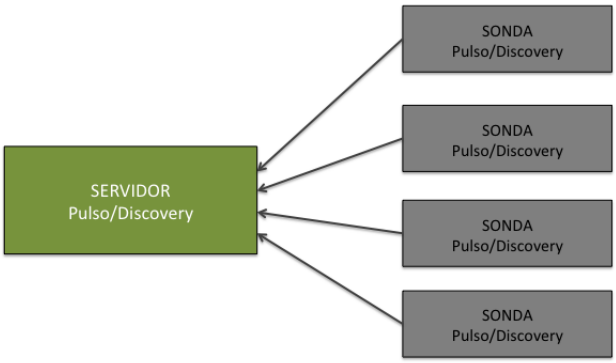

Figura 4.2: Arquitetura base Pulso/Discovery.

4.3 Dados disponibilizados pelas sondas de rede

A recolha de dados é assunto de um outro projeto realizado paralelamente a este na PT Comunicações, pelo que apenas irei fazer referência às ferramentas e ao tipo de dados disponibilizado por cada uma delas e ao modo como estes dados são recebidos por forma a alimentar este projeto.

São usadas três ferramentas de análise de tráfego que fornecem dados ao sistema Pulso/Discovery. Por análise passiva recorre-se às ferramentas P0F[1] e IPAudit[2] e por análise ativa à ferramenta NMAP[3]. Segue-se uma breve descrição de cada uma, para que se possa conhecer que tipo de dados este projeto se baseia para cumprir com os seus objetivos:

Pof Ferramenta utilizada para a identificação de sistemas operativos de um determinado *host*, examinando os pacotes transmitidos a partir ou com destino neste. A análise

[1] `http://lcamtuf.coredump.cx/p0f3`
[2] `http://ipaudit.sourceforge.net/`
[3] `http://nmap.org/`

dos pacotes capturados é realizada ao nível dos protocolos de rede (IP) e transporte, são procuradas assinaturas/*fingerprints* que identifiquem o sistema operativo do recetor/emissor.

IP Audit Ferramenta de monitorização de tráfego de rede. Utiliza uma interface de rede em modo promíscuo e guarda todas as conexões entre dois endereços IP. Fornece dados como endereço de origem e fim (IP/MAC *address*), protocolos e portas de comunicação utilizadas, *bytes*/pacotes enviados/recebidos e *timestamps* de início e fim de comunicação.

Nmap Ferramenta usada para detetar máquinas e serviços na rede, identificando portas e serviços abertos. Fornece dados de serviços tais como, nome, versão, portas e protocolos e dados relativamente aos dispositivos como o tipo e o vendedor do dispositivo, o sistema operativo, endereços de IP e MAC, *hostnames*, entre outros, com as respetivos graus de certeza da ferramenta.

Estas três ferramentas irão alimentar o servidor, procurando satisfazer as necessidades de informação do sistema Pulso/Discovery desenvolvido neste projeto, fornecendo grandes quantidades dados. Dados que terão de ser tratados e armazenados numa base de dados para que possam ser eficazmente geridos.

Futuramente mais ferramentas poderão ser integradas com o objetivo de enriquecer e possibilitar novas funcionalidades à plataforma (como explicado mais à frente na secção 8.3), facto tem de ser levado em conta no desenho da solução.

4.4 Receção e tratamento de dados

A receção e tratamentos dos dados provenientes das ferramentas Pof, Nmap e IP Audit (descritas na secção anterior) é um tema complexo, uma vez que este tipo de ferramentas gera volumes enormes de dados quando em parceria com redes empresarias extremamente complexas e com milhares de dispositivos ligados. O desenho da solução terá que considerar este facto e ser o mais eficiente possível no tratamento destes dados.

As ferramentas apresentadas, embora sirvam propósitos diferentes, muitas vezes capturam dados duplicados ou mesmo contraditórios. Estes problemas de ambiguidade surgem e precisam de ser resolvidos no servidor central, estabelecendo prioridades a certos tipos de dados de certas ferramentas e agregando informação em termos probabilísticos por forma a produzir informação o mais correta possível.

Para além das sondas Pulso/Discovery, outras fontes de dados existentes tornar-se-ão úteis neste projeto, nomeadamente fontes que disponibilizem dados que relacionem ativos de rede conhecidos, tais como aplicações internas existentes, *clusters*, subsistemas ou padrões de comunicação conhecidos, entre outras formas de agregação de ativos/relações.

Existem várias fontes de dados que podem vir a contribuir para enriquecer a plataforma Pulso/Discovery no futuro. É necessário preparar a plataforma e o respetivo modelo de dados para integrar novas fontes de dados.

4.5 Armazenamento e consulta de informação

O questão do armazenamento dos dados terá que ser bem analisado e ponderado uma vez que um dos requisitos funcionais mais importantes e centrais a todo projeto consiste em disponibilizar visualizações úteis e completas sobre os conjuntos de ativos que compõem a rede e as suas respetivas propriedades de vizinhança.

Para obter tais visualizações é necessário realizar consultas que resultem numa grande quantidade de dados armazenados, terão que ser extremamente eficientes e rápidas e refletir forçosamente uma variada e vasta quantidade de propriedades de rede.

Estas propriedades terão de ser refletidas sobre dados de um conjunto grande de ativos de rede e das suas características (tais como *hostnames*, tipos de ativos, sistemas a que pertencem ou serviços que dispõem) e sobre um conjunto de propriedades derivadas dos fluxos de comunicação existentes (tais como protocolos utilizados, *bytes* transferidos, entre outros).

Essencialmente é necessário "abstrair/resumir" a base de dados, traduzindo-a em propriedades genéricas e assim criar visualizações de alto nível para que o analista/utilizador da plataforma possa perceber o que quer analisar e onde, podendo posteriormente consultar dados concretos e detalhados sobre um determinado ativo ou fluxo de comunicação.

Este tipo de consultas sobre bases de dados relacionais e fortemente estruturadas pode tornar-se um problema à medida que a base de dados cresce. O modelo de dados utilizado em versões anteriores do Pulso/Discovery (secção 4.1) poderia servir este projeto mas existem de facto alternativas NoSQL (secção 3.5.1) que melhor se ajustam.

4.6 Visualização e interação com a informação

Tal como descrito anteriormente na secção 1.2, é importante ter um sistema interativo de monitorização e visualização dos elementos que compõem a rede, assim como os seus padrões de comunicação.

A rede é composta por uma enorme diversidade de dispositivos, uns conhecidos e bem identificados pela CMDB (secção 2.2) ou por outros sistemas de monitorização como o Pulso (secção 2.1), e outros desconhecidos que não se encontram identificados pelas mais diversas razões, entre elas porque não participam ativamente em processos importantes ou porque estiveram ligados à rede durante um curto espaço de tempo.

Todos os dispositivos e todas as relações de comunicação são importantes para daí deduzir informação útil, derivar problemas de segurança e ter uma visão global dos fluxos

de dados e dispositivos existentes. É portanto necessário que a partir de dados capturados ativamente e passivamente pelas diferentes ferramentas existentes de análise de tráfego, seja possível tratá-los, relacioná-los e agrupá-los entre si por forma a alimentar uma interface pessoa-máquina interativa que disponibilize toda a informação possível que se possa derivar dos dados.

A aplicação *front-end* deve ser desenvolvida para um ambiente *web*, preferencialmente recorrendo a uma *framework* como o Ruby on Rails[1], dado as vantagens em termos de produtividade e desenvolvimento oferecidas.

Os grafos de rede devem recorrer a uma biblioteca gráfica *web*, o mais eficiente e personalizável possível, utilizando algoritmos *force directed* (secção 3.4), por questões de posicionamento e disposição dos elementos do grafo, de modo a que todos os elementos se mostrem visíveis. Pretende-se que estes grafos de rede sejam gerados rapidamente e que reflitam consultas muito abrangentes como já referido na secção 4.5 anterior.

[1] `http://rubyonrails.org`

Capítulo 5

Desenho da solução

Após uma análise dos objetivos propostos e tendo em consideração o estudo realizado no capítulo do trabalho relacionado (capítulo 3), há a necessidade de desenhar uma solução possível de ser implementada que cumpra todos os requisitos e solucione os problemas analisados no capítulo anterior.

5.1 Receção e tratamento de dados

Uma vez que as sondas, responsáveis pela captura dos dados, são normalmente computadores com pouca capacidade computacional e geralmente se encontram sobrecarregados com outros processos de monitorização de diversos projetos, todo o tratamento dos dados capturados e respetivo cruzamento destes para um posterior armazenamento numa base de dados, terá que ser realizado no servidor central, tal como vem acontecendo em versões anteriores da plataforma Pulso/Discovery (secção 4.1).

Os dados serão recebidos no servidor e validados antes de inseridos na base de dados. Como estes dados provêm de diferentes ferramentas que produzem diferentes resultados, irão ser implementados um conjunto de *parsers* para tratar eficientemente as mensagens recebidas. Os dados serão posteriormente analisados e utilizados para enriquecer a base de dados, caso não representem informação duplicada, desatualizada ou ambígua e que irão ser resolvidas consoante a ferramenta que os gera, o tipo de dados fornecido, segundo graus de certeza ou *timestamps*.

5.2 Neo4j como base de dados

O armazenamento de dados recorrerá a uma solução de modelo de dados NoSQL (secção 3.5.1) por questões de eficiência e escalabilidade, entre outras vantagens estudadas e descritas de seguida (secção 5.3). A distribuição escolhida foi o Neo4j[1].

[1] `http://neo4j.org/`

O Neo4j é uma base de dados orientada a grafos *open source* e suportada comercialmente. Foi lançada a primeira versão em 2010. O acesso à estrutura de armazenamento tais como leituras, escritas, e cruzamentos, são geridos por um sistema de transações ACID, altamente escalável até vários biliões de nós, relações ou propriedades com uma performance aceitável com cerca de 2 milhões de leituras de relações por segundo e com caminhos mais curtos entre estes fornecendo mais escalabilidade quando comparado com um modelo de dados relacional como o MySQL [36].

O Neo4j vem solucionar o problema de queda de desempenho entre consultas que envolvem muitos "*joins*" num RDBMS(*Relational Database Management System*), problema que se iria verificar caso se optasse pela manutenção da base de dados relacional SQL existente em versões anteriores do Pulso/Discovery.

Com a representação de dados recorrendo a grafos, o Neo4j consegue navegar entre os nós e relações com uma velocidade constante, independentemente da quantidade de dados que o constituem. Isto permite algoritmos de grafos muito rápidos, sistemas de recomendação e um estilo OLAP (*On-line Analytical Processing*) de análise que atualmente não é possível com configurações normais em RDBMS.

A navegação pelo grafo é realizada através de algumas API's da ferramenta e o suporte a índices é oferecido via integração com o Apache Lucene[1].

O Neo4j faz parte do movimento NoSQL onde se procura resolver problemas que os RDBMS ainda não conseguiram, relacionados com a complexidade crescente dos dados, as consultas e análises profundas, a escalabilidade e o *sharding* [5, 33, 34].

5.3 Vantagens Neo4j no Pulso/Discovery

As razões que levaram a optar pela base de dados orientada a grafos Neo4j para este projeto são as seguintes:

- Modelação: o uso de uma linguagem de grafos (nós, propriedades e relações) para descrever o domínio tal como se acontece em representações de redes de computadores e respetivos fluxos de comunicação;

- Menos estruturas e maior eficiência de armazenamento de informação semiestruturada, procurando resolver o problema analisado na secção 4.4. Permite mais eficazmente dispor os dados em diferentes níveis de abstração e de profundidade e facilita a integração de novos dados provenientes de outras fontes que futuramente poderão vir a alimentar a plataforma;

- A informação da base de dados é mapeada diretamente para uma linguagem orientada a objetos como o Ruby/Java;

[1] `http://lucene.apache.org/`

- O desempenho no armazenamento de dados e na consulta de informação são excelentes;

- Não existe uma camada de base de dados, permitindo que a implementação, manutenção e configuração sejam mais fáceis. A base de dados é executada no mesmo processo da aplicação;

- As visitas são realizadas recorrendo a uma API intuitiva ou à linguagem Cypher DSL [36];

- Recorre a visitas ao invés de consultas estruturais SQL, que recorrem a uma grande quantidade de junções de tabelas;

- Detem um maior poder de resolução de problemas relacionados com grafos (motores de recomendação, procura do caminho mais curto, entre outros);

- Recorre a transações ACID com suporte a *rollbacks*;

- Facilita a integração com o Ruby on Rails, tornando-se também compatível com uma posterior integração com o portal Pulso (secção 2.1);

5.4 Neo4j em modo de alta disponibilidade

Neste projeto irá implementar-se a base de dados em modo de alta disponibilidade (HA - *High Availability*). Este modo permite:

- Uma arquitetura de base de dados Neo4j tolerante a falhas, onde algumas bases de dados são configuradas para serem réplicas de uma única base de dados mestre. Isto torna o sistema totalmente funcional permitindo sempre leituras e escritas à base de dados mesmo em caso de falhas de *hardware*;

- Um escalonamento horizontal de leituras à base de dados, permitindo ao sistema lidar com uma a carga de leitura maior do que aquela que uma única instância da base de dados pode oferecer.

Para além das vantagens funcionais que as bases de dados Neo4j em modo de alta disponibilidade oferecem, há outras vantagens arquiteturais que foram tidas em conta para a escolha desta funcionalidade para o projeto, nomeadamente a possibilidade de integração com o Ruby on Rails para o desenvolvimento do portal *web* através da respetiva biblioteca Neo4j.rb explicada mais à frente na secção 5.6.

5.4.1 Arquitetura de alta disponibilidade

Quando se recorre ao Neo4j em modo de alta disponibilidade, existe sempre um único mestre/líder e zero ou mais *slaves*/réplicas da base de dados.

São permitidas escritas em réplicas, não sendo preciso redireciona-las para o líder. Uma réplica irá gerir as suas escritas, sincronizando-se com o líder para preservar a consistência do *cluster*, as atualizações são propagadas do líder para as restantes réplicas. Uma escrita numa das réplicas não é imediatamente visível em todas as outras sendo esta a única diferença entre uma única máquina e múltiplas máquinas a correr em modo HA. Todas as outras características ACID mantêm-se.

O Neo4j em modo HA pode ser configurado para diferentes necessidades de carga, tolerância a faltas e disponibilidades de *hardware*. Dentro do cluster, este modo usa o Apache ZooKeeper[1] para eleição de líder e propagação de informação genérica, funcionando como um serviço de coordenação [20].

A base de dados, requer um destes serviços para a eleição de líder inicial, para a eleição de líder em caso de falha do mestre e para publicar informação genérica sobre o *cluster* (por exemplo quando um servidor entra ou sai). Operações de leitura realizadas funcionarão sempre e mesmo as escritas podem sobreviver a falhas de um coordenador se o mestre estiver presente.

O ZooKeeper requer que a maioria das instâncias estejam disponíveis para poder operar corretamente. Isto significa que o número de instâncias ZooKeeper deve ser sempre um número ímpar para garantir todas as suas propriedades e fazer o melhor uso do *hardware* disponível.

5.4.2 Modo de funcionamento

O *cluster* Neo4j HA opera cooperativamente e ativamente através do serviço Apache ZooKeeper. Cada instância irá conectar-se ao serviço, perguntar quem é o líder e ligar-se a este como réplica. Se esta for a primeira a registar-se, torna-se automaticamente líder de acordo com o algoritmo de eleição de líder.

Quando é realizada uma transação numa das réplicas, cada operação de escrita é sincronizada com o líder (*locks* são necessários tanto no líder como na réplica). A transação é primeiramente executada no líder e se esta for *commited* a transação será *commited* também na réplica. Para assegurar a consistência, a réplica tem de estar atualizada com o líder antes de efetuar a operação. Isto é definido no protocolo de comunicação *slave-master* para que as atualizações aconteçam automaticamente, se necessário.

Sempre que um servidor que corre uma base de dados Neo4j fica indisponível, o serviço de coordenação irá detetar e retira-lo do *cluster*. Se o líder ficar indisponível, um novo líder será eleito automaticamente. Normalmente um novo líder é eleito e inici-

[1] `http://zookeeper.apache.org/`

ado dentro de poucos segundos e durante este tempo, escritas não podem ser realizadas. Quando a máquina volta a estar disponível irá automaticamente ligar-se ao *cluster*.

A única ocasião em que isto não se verifica é quando um líder antigo realiza alterações que nunca chegaram a ser replicadas para as restantes máquinas. Se o novo líder eleito efetuar alterações antes do líder antigo recuperar, haverá duas versões diferentes da base de dados. O líder antigo remove a sua base de dados e obtém uma cópia da do novo líder. Resumindo [36]:

- As réplicas podem realizar transações de escrita;
- As atualizações para as réplicas são *eventually* consistentes;
- O Neo4j em modo de alta dispobilidade é tolerante a faltas (dependendo da configuração ZooKeeper);
- As réplicas serão automaticamente sincronizadas com o líder nas operações de escrita;
- Se o líder falha um novo líder será eleito automaticamente;
- As máquinas reconectam-se automaticamente ao *cluster*, sempre que haja um problema de rede/manutenção/*timeout*;
- As transações são atómicas;
- Se o líder ficar indisponível, qualquer transação de escrita ativa será revertida e durante a eleição de líder nenhuma escrita poderá ser realizada;
- Leituras são altamente disponíveis.

5.5 Portal web Pulso/Discovery

Para a implementação do novo portal Pulso/Discovery, recorrer-se-á à mais recente versão da *Framework* Ruby on Rails e respetivas bibliotecas.

Para a construção dos grafos de comunicação dos elementos que compõem a rede irá utilizar-se a biblioteca D3.js[1] para que seja possível interagir com os elementos, e perceber a nuvem de comunicação, protocolos usados, volumes de tráfego, tipos de dispositivos, etc. que caracterizam a rede.

O D3.js é uma biblioteca para manipulação de documentos baseados em dados. Esta ajuda a transformar dados em informação usando HTML, SVG e CSS. A enfâse desta biblioteca é nos *standards web*, o que permite tirar partido de todas as capacidades dos

[1] `http://d3js.org`

browsers modernos, sem estar limitado às suas funcionalidades. Permite personalizações diversas e independentes da sua API.

Combina visualizações eficazes e inovadoras através uma abordagem orientada aos dados para a manipulação do DOM. Esta ferramenta encontra-se em constante crescimento e oferece uma crescente game de visualizações úteis.

Esta biblioteca oferece implementações de visualizações *force directed* (secção 5.5).

5.5.1 D3 *force layout*

A componente *force layout* do D3.js é utilizada para disponibilizar a maioria das funcionalidades por detrás das transições, animações e colisões nas suas visualizações.

Os componentes que compõem a visualização atraem-se e repulsam-se à medida que atingem as suas posições finais. Este tipo de simulação física nas visualizações são tipicamente utilizadas para criar visualizações de grafos. Não é necessário ter um conhecimento profundo acerca de teoria de grafos para perceber como estas visualizações funcionam, no entanto, é necessário conhecer as variáveis que controlam este tipo de visualizações e saber como as utilizar.

Algumas das mais importantes variáveis físicas a que este tipo de visualizações recorrem, são:

Gravidade A gravidade representa uma força que atrai os nós para o centro do grafo. Quanto mais perto está o nó, menos impacto tem o parâmetro de gravidade neste. A gravidade pode também ser negativa, fazendo com que os nós sejam empurrados para longe do centro.

Fricção Em cada iteração da simulação, o movimento dos nós são escalados por este valor (limitado entre zero e um), sendo que o valor zero representa ausência de movimento e o valor um, a ausência de fricção.

Repulsão Este atributo refere-se à repulsão ou atração entre elementos, nomeadamente nós nesta biblioteca. Tal como um íman, os nós podem ter uma força negativa (repulsão) ou positiva (atração).

Alfa Descrito como um "factor de arrefecimento". O alfa é utilizado para escalar o movimento dos nós na simulação para que no início estes se movam relativamente depressa e para que no fim, os nós mal se movam das suas posições. Possibilita-se um amortecimento dos efeitos das forças na visualização ao longo do tempo. Sem o alfa, atributos como a gravidade iriam exercer uma força constante sobre os nós e a simulação tornava-se confusa e instável.

5.6 Portal *web* e interação com o Neo4j

No portal *web*, através do Ruby on Rails, a aplicação vai recorrer ao Neo4j.rb, um *wrapper* JRuby para bases de dados orientadas a grafos Neo4j. A biblioteca Neo4j.rb é composta por uma API de 3 camadas:

Layer 1 (Neo4j-core) Possibilita a interação direta com os elementos básicos da base de dados (nós, relações e propriedades).

Layer 2 (Neo4j-wrapper) Possibilita o encapsulamento de objetos de *layer* 1.

Layer 1 (Neo4j) Disponibiliza implementações para o Rails Active Model e um subconjunto da API Active Record Rails

Com este *wrapper* é possível mapear dados provenientes da base de dados Neo4j em Modelos Rails de uma forma quase automática, tirando todo o partido do Active Model e Active Record da ferramenta. A implementação da aplicação *web* fica assim mais organizada e a sua manutenção é facilitada.

5.7 Arquitetura Pulso/Discovery

A arquitetura Pulso/Discovery (figura 5.1) irá basear-se no *cluster* de base de dados Neo4j em modo de alta disponibilidade, explicado na secção 5.4.

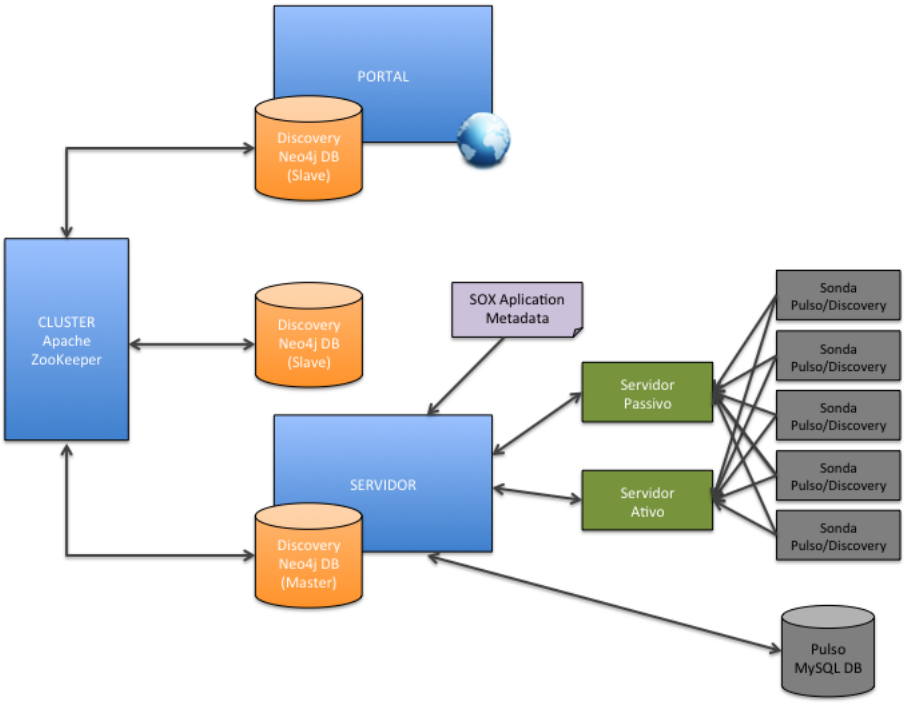

Figura 5.1: Arquitetura do novo Pulso/Discovery

A usar as instâncias da base de dados Neo4j iremos ter por um lado o servidor central, que por questões de eficiência irá trabalhar sobre a instância *master* (pese necessidade de uma grande quantidade de escritas e consultas) e por outro, o portal *web*, que interage igualmente com a base de dados Neo4j, mas na grande maioria dos casos através de operações de leitura.

Para além dos dados fornecidos pelas sondas de rede Pulso/Discovery, o servidor central irá também aceder a outras fontes de dados externos á plataforma. Estas fontes são a base de dados Pulso e os ficheiros de metadados de aplicações conhecidas, nomeadamente aplicações SOX (secção 2.3).

5.8 Esboço das interfaces

Foram ponderados vários modelos e desenhados alguns esboços para perceber a melhor forma de conceber uma interface pessoa-máquina para o novo portal Pulso/Discovery.

Os esboços foram desenhados recorrendo a uma ferramenta básica de desenho. Estes foram estudados e ponderados junto dos elementos da equipa responsável pelo projeto Pulso/Discovery e iterativamente melhorados a partir dos que são apresentados nas figuras 5.2, 5.3 e 5.4, tendo sido ponderados um conjunto de funcionalidades e interações possíveis, consoante com os dados disponíveis.

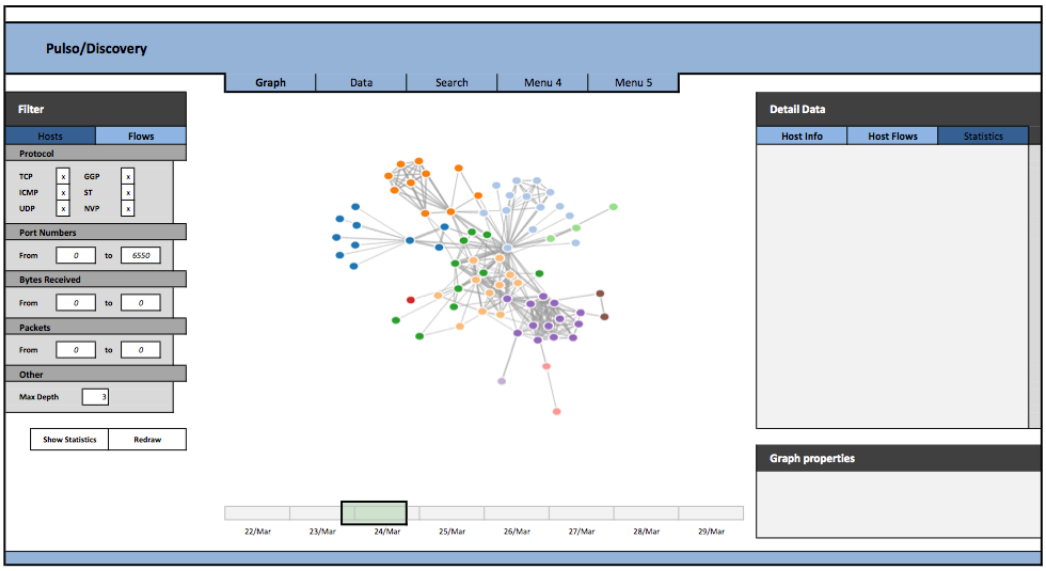

Figura 5.2: Esboço da interface pessoa-máquina, *layout*.

Estes esboços foram sendo refinados à medida que o projeto evoluiu dando origem à interface pessoa *web* apresentada mais à frente na secção 6.3.4.

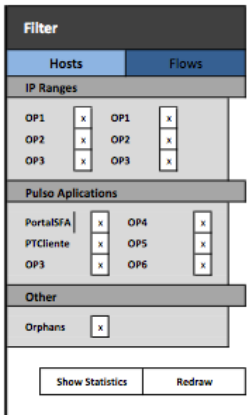

Figura 5.3: Esboço da interface pessoa-máquina, filtros.

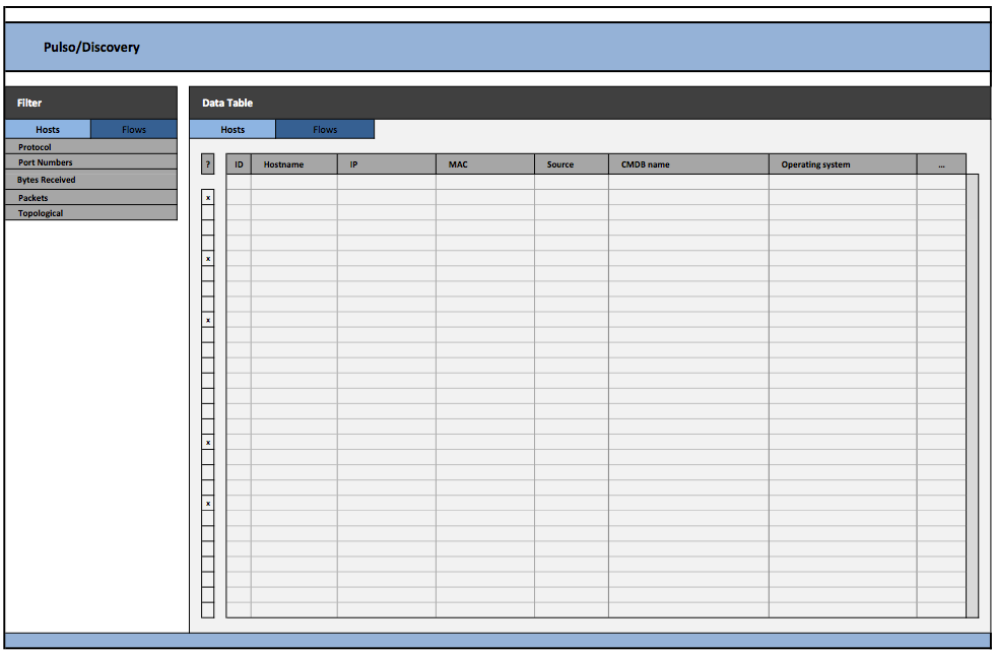

Figura 5.4: Esboço da interface pessoa-máquina, informação detalhada.

Capítulo 6

Implementação

Este capítulo descreve a implementação da solução, tendo em consideração os requisitos funcionais e técnicos identificados pelo modelo teórico, seguindo a arquitetura estudada no capítulo anterior. Foram definidas três aplicações distintas que subdividem a arquitetura da forma como apresenta a figura 6.1.

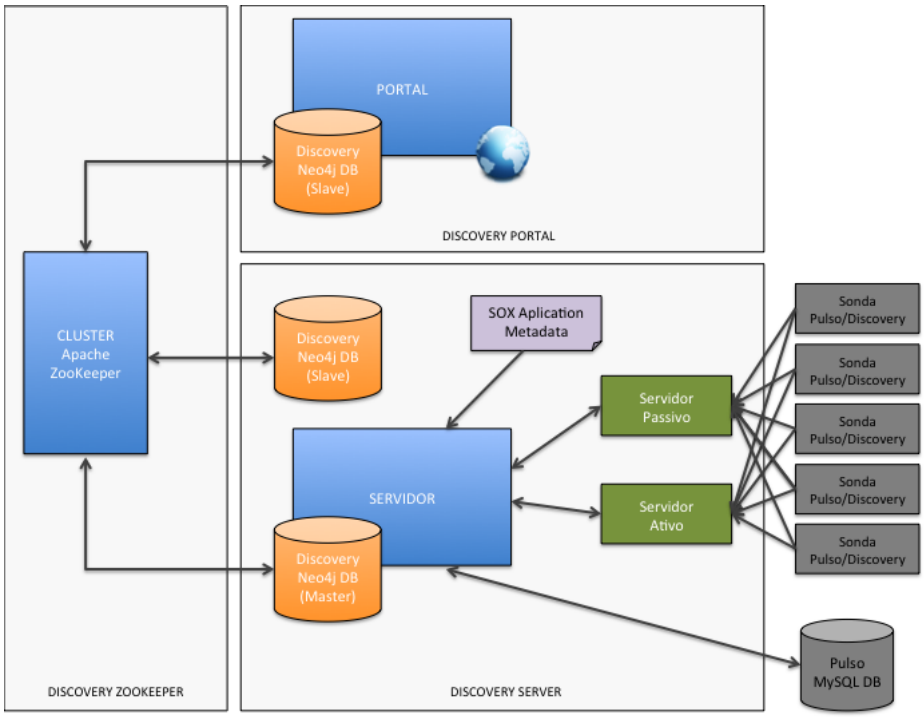

Figura 6.1: Arquitetura Pulso/Discovery subdividida em três módulos distintos: DiscoveryZookeeper, DiscoveryServer e DiscoveryPortal.

As três módulos são os seguintes:

DiscoveryZookeeper Responsável por gerir o *cluster* de alta disponibilidade Neo4j, criando e configurando as suas instancias e as respetivas bases de dados.

DiscoveryServer Responsável pela comunicação com as sondas, e pelo tratamento dos

dados fornecidos. Recorre também a outras fontes de informação externas, uma base de dados relacional e metadados de aplicações para alimentar o repositório Neo4j com informação relevante.

DiscoveryPortal Responsável por disponibilizar uma interface pessoa-máquina capaz de responder a uma série de questões de segurança relevantes, possibilitando controlar a existência ou não de relações de comunicações na rede entre ativos pertencentes ou não a aplicações críticas aos processos de negócio da PT Comunicações.

6.1 DiscoveryZookeeper

Este módulo consiste numa pequena aplicação escrita em Ruby, capaz de configurar e gerir todo um sistema de *clustering* de bases de dados Neo4j. É responsável por instalar as diferentes bases de dados nas diferentes máquinas e configurar o *cluster*. *Cluster* que é composto por três máquinas (no mínimo) ou máquinas virtuais e um coordenador em cada uma, onde é executada uma instância Neo4j de alta disponibilidade (HA).

Um *cluster* de três máquinas é uma configuração conservadora em termos de uso de *hardware*, é capaz de lidar com uma carga de leitura moderada quando comparado com *clusters* de maiores dimensões.

Esta aplicação disponibiliza um conjunto de operações necessárias para interagir com o *cluster* e com as suas bases de dados Neo4j.

6.1.1 Operações DiscoveryZookeeper

As operações disponibilizadas por este modo são as seguintes:

Create Cluster Faz o download da base de dados Neo4j (versão *enterprise 1.7*) e instala-a em cada uma das máquinas do *cluster*. São configuradas as diferentes instâncias para poderem colaborar entre si, sendo que para isso são definidas as seguintes propriedades para cada um dos elementos do *cluster*:

- O ID para a instância Neo4j, definindo-a como elemento de um *cluster* Neo4j HA;
- O IP e porta para a comunicação para com os restantes elementos;
- A lista de coordenadores dos servidores que compõem o *cluster* (*hosts* e portas);
- O intervalo de tempo para sincronização de dados com o líder;
- O nome do *cluster* (útil para o caso de haver vários clusters Neo4j HA geridos pelo mesmo coordenador);

- A porta HTTP para acesso aos dados administrativos, e conteúdo da base de dados através de uma interface *web* disponibilizada pela ferramenta;
- O diretório de localização da base de dados de cada elemento;
- A porta JMX para cada uma das instâncias;
- O conjunto de portas de comunicação internas do *cluster*, *quorum election port* e *leader election port*.

Delete Cluster Apaga as várias instâncias do *cluster* e as suas bases de dados.

Start coordinators Arranca os coordenadores de cada máquina que compõem o *cluster*.

Stop coordinators Pára todos os coordenadores do *cluster* que se encontram ativos.

Restart coordinators Reinicia todos os coordenadores que compõem o *cluster*.

Reset databases Apaga e reinicia todas as bases de dados que compõem o *cluster*.

Start webadmin Disponibiliza os dados e as configurações da base de dados, através de uma interface *web* fornecida pela distribuição Neo4j.

Stop webadmin Pára a instância *webadmin*, caso esteja a ser executada.

6.1.2 Arquitetura

A arquitetura de alta disponibilidade Neo4j para o projeto Pulso/Discovery é apresentada na figura 6.2.

É composta por pelo menos três máquinas/máquinas virtuais, onde a segunda instância é definida como a base de dados mestre, e as restantes como réplicas.

6.1.3 Instâncias da base de dados Neo4j

São criadas, por predefinição, três instâncias da base de dados. Denominadas como Neo4jMaster/Embedded, Neo4jRailsSlave e Neo4jReplicaSlave.

A instância Neo4jMaster é usada pelo módulo DiscoveryServer (secção 6.2), onde irão ser realizadas a maior parte das escritas e grande parte das consultas. As restantes serão réplicas, sincronizando-se periodicamente com esta. A instância Neo4jRailsSlave é usada pela aplicação *front-end* DiscoveryPortal (secção 6.3).

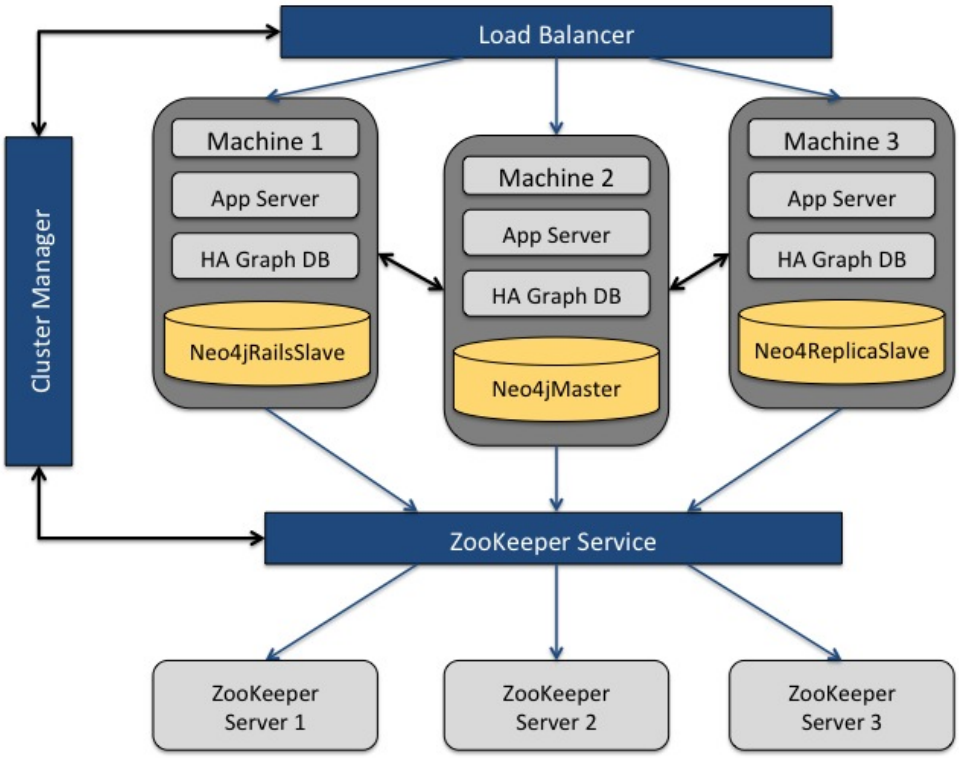

Figura 6.2: Arquitetura DiscoveryZookeeper, composto por um *cluster* Neo4j HA com três máquinas.

6.2 DiscoveryServer

Este é o módulo central a todo o projeto. É aqui que é implementada a comunicação e recolha de dados das sondas Pulso/Discovery e a consulta à base de dados relacional Pulso e aos metadados de aplicações importantes a monitorizar. É aqui também que se processa o tratamento de dados e gestão do repositório Neo4j para posteriormente ser consultado pela aplicação *front-end*.

6.2.1 Arquitetura

Este módulo do projeto conta com dois submódulos responsáveis pela comunicação sonda/servidor, um por cada tipo de dados: módulo passivo (para as ferramentas P0F e IPAudit) e módulo ativo (para a ferramenta NMAP), cada um responsável pela respetiva receção de dados, *parsing*, tratamento e inserção da informação no repositório. São definidas diferentes portas de comunicação para cada um dos tipos de dados recebidos.

Esta aplicação conta com dois servidores que aguardam dados provenientes das sondas. Cada vez que é recebida uma mensagem, é lançado uma *thread* responsável pelo tratamento dos dados, deixando a *thread* principal livre para receber novas requisições.

Para além dos dados das sondas Pulso/Discovery, a base de dados Neo4j é alimentada por metadados de aplicações, subsistemas e *clusters* (figura 6.3) por forma a perceber que *hosts* as constituem e representando essa informação em árvore na base de dados Neo4j, como explicado mais a frente na secção 6.2.3.

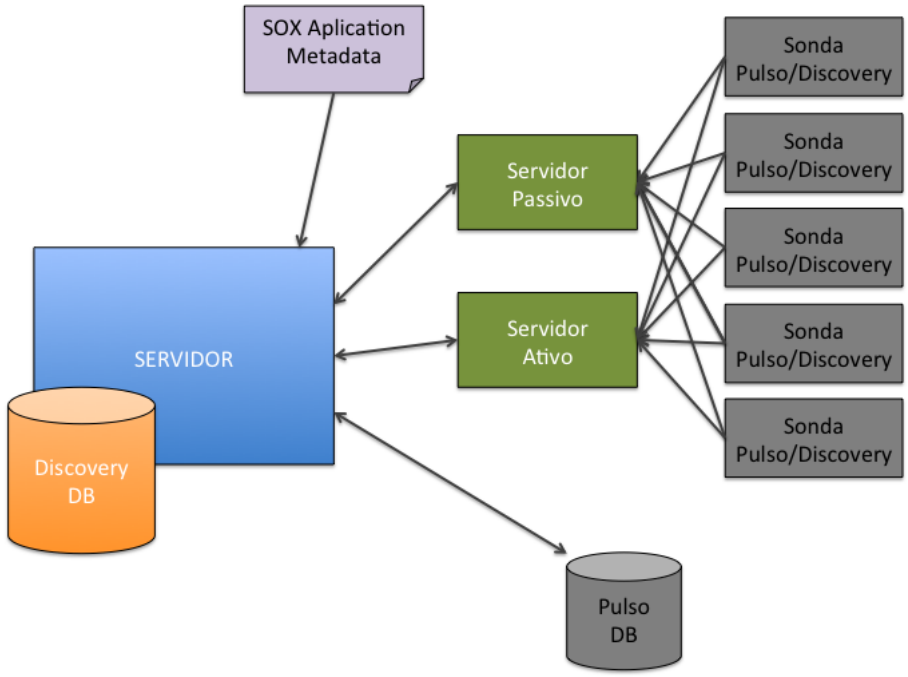

Figura 6.3: Arquitetura DiscoveryServer

6.2.2 Processamento de dados

Este módulo faz todo um pré-processamento de dados e metadados essenciais à aplicação para uma posterior inserção na base de dados Neo4j. Os dados provêm de diversas fontes: ferramentas (Nmap, IP Audit e P0F), base de dados relacional Pulso e ficheiros de metadados de aplicações SOX.

Estes dados são validados e confrontados com os já existentes na base de dados, percebendo-se assim se contribuem ou não para o enriquecimento da informação.

Fora do âmbito deste projeto, que se limita a receber as mensagens e a tratar os dados, cada mensagem enviada pelas sondas é composta por um conjunto muito grande blocos, onde cada bloco é constituído por vários campos. O tamanho destas mensagens é definido na sonda que realiza o envio.

Os campos que compõem os blocos e sub-blocos recebidos das pelos dois servidores são seguintes:

- Servidor ativo (NMAP):

  - `IP; MAC address; CPU Vendor; Hostname; Operation System; Operation System Accuracy;`
    * `Protocol, Port, Service Name; Service Version; Service Extra Info; Service Accuracy;`

- Servidor passivo (IPAudit e P0F):
  - `Host1 IP; Host1 MAC Address; Host1 Port; Host1 Packets; Host1 Bytes; Host2 IP; Host2 MAC Address; Host2 Port; Host2 Packets; Host2 Bytes; Protocol; Time First Packet; Time Last Packet; Who Sent First; Who Sent Last; Host1 Operation System; Host2 Operation System;`

Os dados referentes a fluxos de comunicação entre ativos são fornecidos pela ferramenta IPAudit. Para além das propriedades de comunicação, nomeadamente protocolos utilizados, quantidades de *bytes* e pacotes transferidos, etiquetas temporais e direção dos fluxos, é possível derivar novos ativos através do par de atributos `IP - MAC` *`address`*.

O NMAP fornece propriedades e serviços (e respetivos graus de certeza) de um dado ativo, propriedades que apenas uma análise de tráfego ativa consegue derivar. O P0F complementa esta informação ao derivar os sistemas operativos dos diferente hosts comunicantes.

Estes dados são validados e confrontados com os já existentes. Serão posteriormente armazenados na base de dados caso constituam uma informação nova para o servidor.

Na base de dados um ativo é representado por um nó e um fluxo de comunicação uma relação entre dois nós, com a direção definida pelo atributo `Who Sent First`.

6.2.3 Metadados de aplicações

Para se poder relacionar ativos encontrados ou relações de comunicação com algumas aplicações existentes na rede empresarial, é necessário consultar metadados existentes na base de dados MySQL Pulso e consultar ficheiros que descrevem os ativos responsáveis pelas aplicações SOX, uma vez que estes não se encontram sobre monitorização Pulso.

A partir da base de dados Pulso são consultados dados referentes a estruturas de aplicações, subsistemas e *clusters*, assim como os respetivos ativos. De ficheiros, retira-se os ativos pertencentes às aplicações SOX. Todos estes metadados de aplicações são armazenados numa estrutura em árvore na base de dados Neo4j por forma a facilitar as posteriores consultas.

Esta cache de informação que é mantida na base de dados é atualizada manualmente sempre que se mostre necessário, bastando dar a indicação à ferramenta e todo o processo é automático.

6.2.4 Base de dados

No módulo DiscoveryServer, há a opção de recorrer a uma de duas técnicas de inserção na base de dados Neo4j. Estas recorrem respetivamente à API embebida e à API REST [36]. Esta opção é definida no ficheiro de configuração da aplicação (secção 6.2.5) e será discutido o uso de ambas no capítulo 8.1.

Um grafo Neo4j é (capítulo 3.5.1) composto por:

- Nós que são ligados por
- Relações, com
- Propriedades em ambos os nós e relações.

Todas as relações Neo4j têm um tipo. Por exemplo, o grafo que representa a comunicação entre dois ativos A e B é representado pela relação "*Flow*". Se uma relação do tipo "*Flow*" interliga dois nós, isto significa que os dois ativos comunicaram, onde a direção da relação indica quem a iniciou (distinguindo-se assim o emissor do recetor).

Muito da semântica de um grafo Neo4j é codificado nos tipos das relações que interligam os diferentes nós. Quanto aos nós, as bases de dados Neo4j não definem um tipo tal como acontece com as relações. No entanto, isto foi desenhado tendo como exemplo a forma como o Active Model do Ruby on Rails para bases de dados Neo4j internamente os modela. Isto é implementado ao relacionar todos os nós de um determinado tipo com um nó "mestre". Assim, para obter todos os nós de um determinado tipo é apenas necessário fazer uma visita a partir deste nó de profundidade um, na direcção *outgoing*.

O "modelo" de dados utilizado é o apresentado na figura 6.4, onde é possível distinguir diferentes tipos de nós e relações e percorrer o grafo segundo propriedades relevantes e recorrendo à Traversal API e à indexação Apache Lucene de nós e relações. Note-se que as visitas são realizadas com a mesma eficiência independentemente da direção das relações que se pretenda seguir.

Tendo considerado a integração da base de dados com a ferramenta Ruby on Rails e posterior modelação de nós e relações para classes Ruby através do Active Model, os dados são adicionados à base de dados seguindo o modelo apresentado (figura 6.4). Para que este modelo de dados seja compatível com a aplicação DiscoveryPortal e para que os nós e as relações sejam automaticamente mapeados em objetos Ruby, bastam dois procedimentos, adicionar um atributo "*classname*" às diferentes instâncias, e relacionar estas com o nó genérico da classe que pretende representar.

Operações de escrita na base de dados

Todas as operações de escrita na base de dados são realizadas recorrendo a transações. O uso de transações Neo4j apoia-se em blocos *try-finally*, sendo que a ultima operação

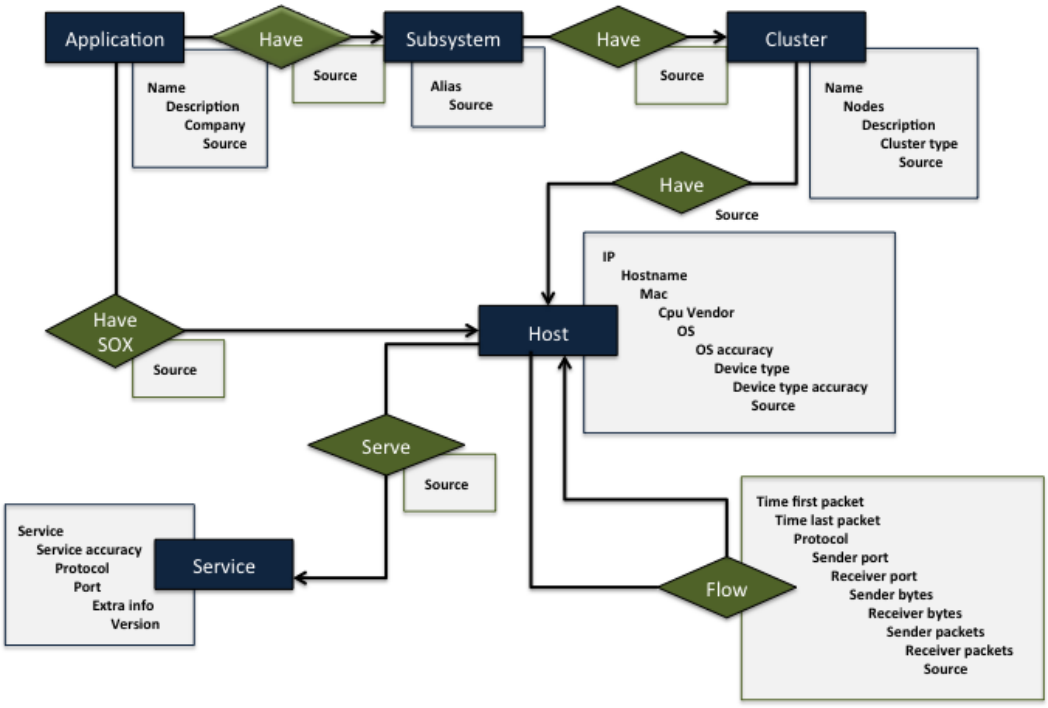

Figura 6.4: "Modelo" da base de dados Neo4j do projeto Pulso/Discovery, representando à azul os nós e a verde as relações entre nós assim como as respetivas propriedades.

deste bloco finaliza a transação, realizando o respectivo *commit* ou *rollback*, dependendo do sucesso, ou não, das operações realizadas [36].

O uso de transações neste projeto possibilita manter a integridade dos dados e garantir um bom comportamento através das propriedades ACID:

Atomicidade Se parte de uma transação falhar, o estado da base de dados mantêm-se inalterado;

Consistência Qualquer transação irá deixar a base de dados num estado consistente;

Isolamento Durante uma transação, os dados modificados não podem ser acedidos por outras operações;

Durabilidade O DBMS pode sempre recuperar os resultados a partir de uma transação bem sucedida;

6.2.5 Bibliotecas utilizadas

Este módulo é desenvolvido em Java e faz uso das seguintes bibliotecas:

Jackson v1.9.2 Para processamento de mensagens JSON trocadas entre o cliente Java e o *web service* REST Neo4j. Usado para o *parsing* e construção de mensagens JSON.

Jersey v1.11 Para estabelecer a comunicação entre o cliente e o *web service* Neo4j, quer para inserir, quer para consultar informação da base de dados.

Apache Log4j v1.2.16 Biblioteca de *logging* usada tanto para os *outputs* da aplicação DiscoveryServer, como para o Apache ZooKeeper utilizado pelas bases de dados Neo4j.

Mysql conector v5.1.2 Para coneção JDBC e consulta da base de dados relacional Pulso, nomeadamente a estrutura de aplicações, subsistemas, *clusters* e respetivos ativos.

Apache Lucene v3.5 Usado em colaboração com a base de dados Neo4j para indexação de propriedades quer de relações, quer de nós.

Zookeeper v3.3.2 Usado pelo *cluster* Neo4j HA para coordenação dos servidores de bases de dados existentes.

Neo4j v1.7 Biblioteca que permite a interacção com a base de dados orientada a grafos Neo4j, usada como repositório de dados deste projeto.

6.2.6 Configurações

Este módulo do projeto disponibiliza um ficheiro de configurações onde são definidas várias opções de configuração essencias, como mostra o exemplo da figura 6.5.

Define-se ficheiros de *log*, portas de comunicação, dados de acesso à base de dados Pulso, modo de interação com a base de dados Neo4j (REST ou embebido), localização da base de dados e localização do ficheiro de configuração do Apache ZooKeeper para funcionar em modo de alta disponibilidade. Por fim é possível definir a modelação Active Model utilizada, nomeadamente de nós e relações para classes Ruby.

6.3 DiscoveryPortal

Este módulo do projeto é responsável pela visualização e interação com a informação.

Aqui é implementada uma interface pessoa-máquina capaz de fornecer ao utilizador informações sobre a rede, os seus componentes e as suas relações de comunicação através de visualizações interativas e recorrendo a tecnologias *web*.

```
#-----------------------------------------------#
#--     DiscoveryNeo4jServer configuration      --#
#-----------------------------------------------#

#-- Log files --#
server_log_file = log/Server.log
db_log_file = log/DB.log

#-- Server ports --#
ipaudit_port = 6789
nmap_port = 6666

#-- Pulso database --#
db_pulso_name = pub_metadata
db_pulso_host = 10.177.168.137
db_pulso_user = pulso
db_pulso_password = *****

#-- Graph database --#
#db_type = rest
db_type = embedded
db_root_uri = http://localhost:7474/db/data/
db_path = ../DiscoveryZookeeper/Neo4jServer1.Master/data/db/
ha1_conf = conf/ha1-neo4j.properties

#-- Rails active model --#
rails_nodes = Host;Service;Application;Subsystem;Cluster
rails_relationships = Host#flow_to;Host#service_to;Application#have_to;Application#have_sox_to;Subsystem#have_to;Cluster#have_to;_all;
```

Figura 6.5: Exemplo de ficheiro de configuração DiscoveryServer.

6.3.1 Visualização e pesquisa de ativos

A visualização dos elementos que compõem a rede é baseado num sistema de procura por um determinado IP ou gama de endereços de IP's, indicando ou não, quais os elementos de aplicações que se pretende, permitindo juntar os vários ativos encontrados, numa só visualização.

Os resultados são apresentados em forma grafo, através da biblioteca d3.js (secção 5.5). Os grafos são compostos por círculos/nós que representam os ativos de rede encontrados. A cor e o tamanho destes representam a relevância que detêm na rede.

Para dar uma noção de profundidade ao grafo, alguns nós são desenhados com um nível de transparência alto representando ativos que, apesar de não fazerem parte pesquisa, apresentam relações de comunicação com os ativos encontrados. As ligações entre nós representam o tráfego existente entre ativos onde a espessura reflete uma abstração da quantidade de dados envolvidos.

Esta personalização (discutida mais a frente na secção 8.3.3) dos elementos através da cor, tamanho, forma, transparência, entre outros, poderia tornar-se ainda mais rica, embora haja a necessidade de estabelecer um equilíbrio entre o que se pretende mostrar e o que o utilizador pode inferir ou deduzir, isto é, criando uma visualização que não influencie o analista com demasiadas propriedades que possam evidenciar aspetos irrelevantes de analisar.

A interação com os grafos gerados permite uma posterior consulta de informação detalhada, que em parte se encontra abstraída pelos atributos do grafo. A partir dos resultados de uma primeira aproximação, há a possibilidade de excluir ou incluir informação desejada recorrendo a filtros, permitindo iterativamente ajustar o grafo ao que pretende

visualizar.

A figura 6.6, mostra um exemplo de um grafo obtido pelo DiscoveryPortal, onde se pode visualizar e interagir com os ativos encontrados. Em poucos segundos é possível ter uma noção de quais os ativos descobertos numa dada gama de indereços de IP, as suas relações de comunicação com outros ativos pertencentes à mesma gama ou fora, e quais os ativos existentes que pertencem, ou não, a aplicações/subsistemas/*clusters* críticos, assim como os volumes de comunicação trocados entre si.

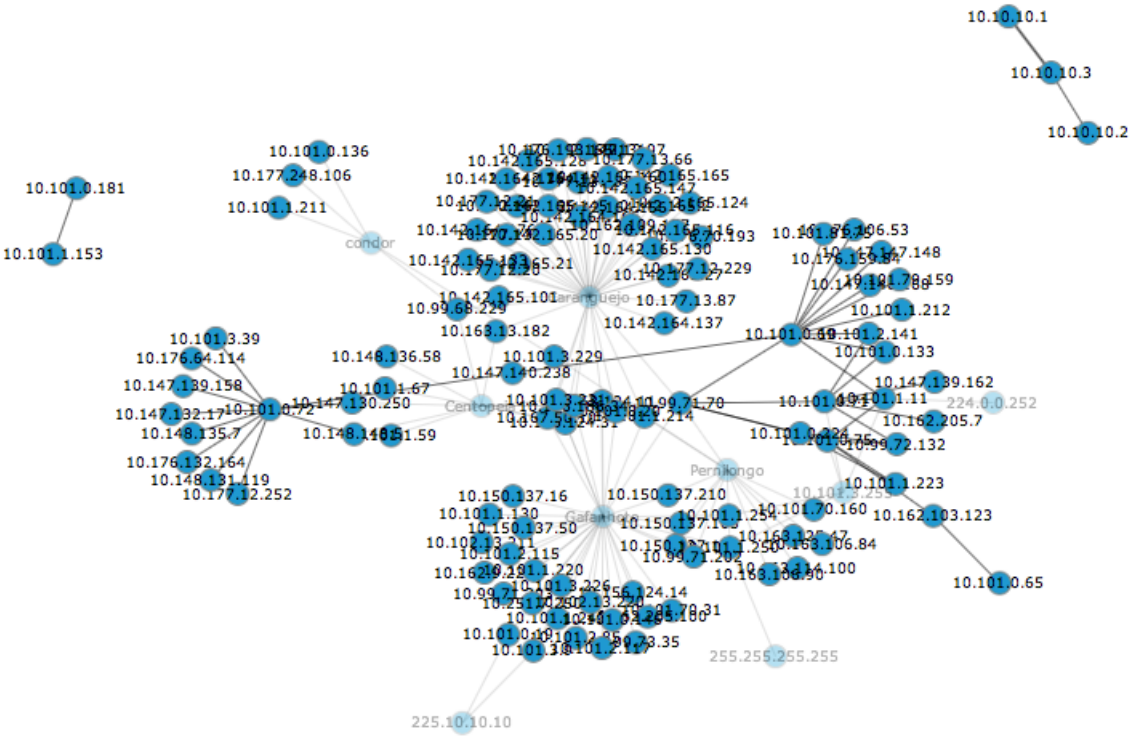

Figura 6.6: Grafo gerado pelo portal Pulso/Discovery, resultado de uma pesquisa efetuada na gama de endereços 10.101.0.0/16 dentro da rede empresarial da PT Comunicações.

6.3.2 Adição e filtragem da informação

Conforme as opções de pesquisa definidas, o grafo de resultados pode tornar-se demasiado complexo, apresentando informação excessiva ou indesejada, ou demasiado simplista, apresentando poucos resultados úteis de analisar.

Estes problemas podem ser resolvidos em grande parte, recorrendo a técnicas de filtragem dos dados, adicionando ou removendo elementos segundo as suas características.

Foi implementado um menu de filtros sobre o grafo de visualização como mostra a figura 6.7.

No caso dos nós é possível ajustar-se a diferentes gamas de endereços de IP, limitando ou adicionando ativos e respetivas comunicações à visualização. É possível: adicionar ou

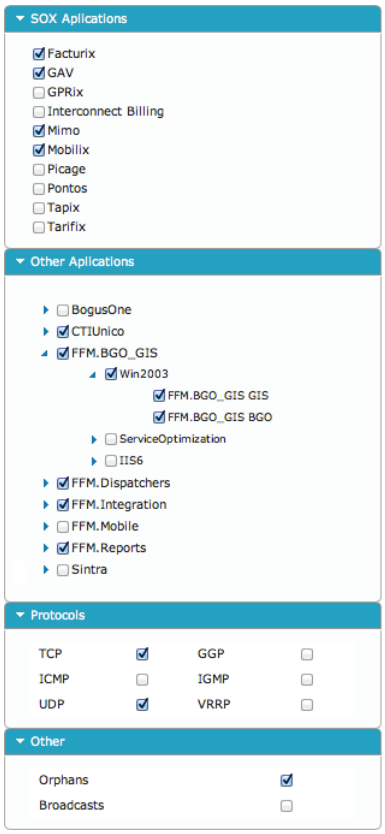

Figura 6.7: Interface para filtragem de nós e ligações visualizadas no grafo de rede

remover elementos de aplicações SOX; adicionar ou remover elementos de aplicações, subsistemas ou *clusters* atualmente monitorizadas pelo sistema Pulso e indicar à ferramenta para excluir ou não, ativos que não apresentem qualquer tipo de comunicação entre os elementos procurados.

Nas ligações de comunicação é possível filtrar por protocolo de rede utilizado, podendo utilizar-se diferentes combinações de protocolos e excluir ou incluir *broadcasts* de rede que por norma causam muito ruído nos grafos, permitindo ao utilizador focar-se em aspetos realmente importantes de analisar. As opções de filtragem definidas pelo utilizador são mantidos em dados sessão e em *cookies* HTTP.

Os filtros enumerados foram ponderados consoante as necessidades e os requisitos funcionais propostos à aplicação sobre os dados que o Pulso/Discovery atualmente tem acesso, sendo que a implementação de mais filtros baseados em outras propriedades existentes mostraram-se pouco úteis e foram retirados após alguns testes com o portal, filtros baseados em propriedades como a quantidade de pacotes ou *bytes* transferidos. Inversamente, outros mostraram-se importantes embora não existissem dados para os definir. Este assunto é discutido em maior profundidade na secção 8.3.

6.3.3 Interação com os grafos

Os grafos de rede apresentados pela ferramenta não se limitam a fornecer informação ao utilizador, é possível interagir com estes por forma a responder a um conjunto de requisitos importantes. A interação com os grafos permite:

- O arrastamento dos nós e ligações para uma posição diferente, facilitando a perceção de certas características do grafo, figura 6.8;
- A visualização de *tooltips* de nós, nomeadamente endereços de IP caso a etiqueta deste indique um *hostname*, figura 6.9;
- Um evento de *click* sobre o nó, que mostrará informação adicional detalhada sobre o ativo. Detalhes das suas relações, serviços e aplicaçoes (secção 6.3.3) como apresenta a figura 6.9;

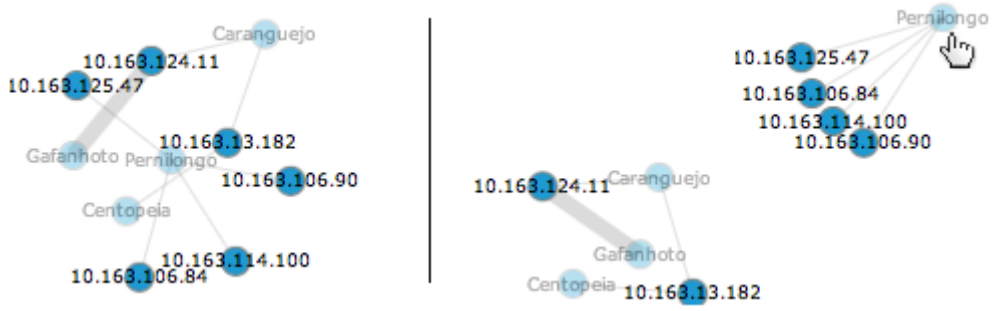

Figura 6.8: Exemplo de arrastamento de um elemento do grafo para uma posição diferente.

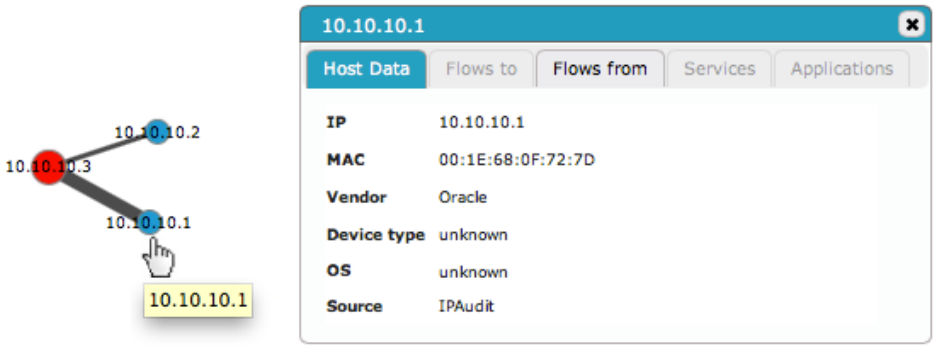

Figura 6.9: Exemplo de uma *tooltip* e de um evento *click* sobre um ativo representado no grafo.

Além destes três modos de interação com o grafo, a ferramenta oferece a configuração de propriedades *force directed*, nomeadamente níveis de repulsão, tamanho das ligações,

fricção e forças (secção 5.5), tornando possível ao utilizador ajustar a disposição dos elementos do grafo consoante os resultados das pesquisas realizadas (figura 6.10). Permite uma visualização mais ajustada a cada caso resolvendo problemas de oclusão ou dispersão de elementos que podem dificultar a análise do grafo.

Para além das funcionalidades implementadas sobre os grafos de rede, à que referir outras possível melhorias e implementações futuras, nomeadamente a técnica *fisheye* descrito em mais pormenor no capítulo 6.3.3.

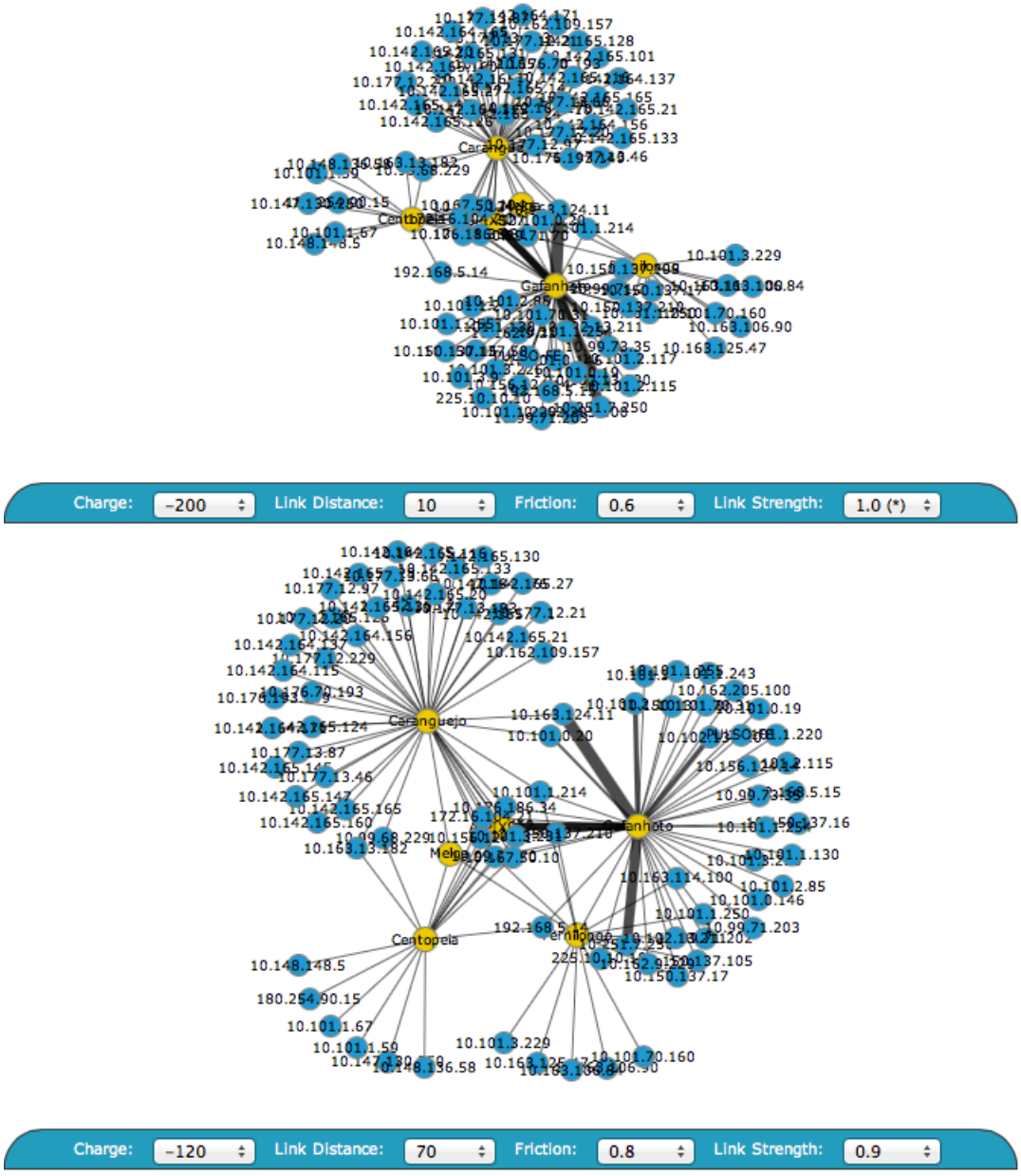

Figura 6.10: Exemplo de ajuste das propriedades físicas do grafo, alterando a disposição dos elementos que o compõem.

6.3.4 Informação sobre ativos

Ao selecionar um nó do grafo é disponibilizado ao utilizador informação adicional sobre o ativo correspondente, assim como os seus fluxos de comunicação.

Esta informação detalhada é realizada recorrendo à tecnologia AJAX, tal como grande parte das funcionalidades do portal Pulso/Discovery. A consulta correspondente é realizada à base de dados em *background* e assim que a resposta é recebida é apresentado ao utilizador uma janela com a informação correspondente.

Esta informação é apresentada em cinco separadores distintos, cada um deles contendo a seguinte informação (figura 6.11):

Host data Características gerais do dispositivo, tais como IP, MAC *address*, *hostname*, vendedor, tipo e sistema operativo, assim como os graus de certeza dos dois últimos e as fontes responsáveis pelos dados apresentados.

Flow to Resumo dos fluxos de comunicação do ativo no papel de emissor, agrupados por protocolos e por *hosts* de destino.

Flow from Resumo dos fluxos de comunicação do ativo no papel de receptor, agrupados por protocolos e por *hosts* de origem.

Services Uma lista resumida de todos os serviços detetados no ativo.

Applications Uma lista das aplicações ou subsistemas que utilizam este *host* como parte fundamental para o seu funcionamento.

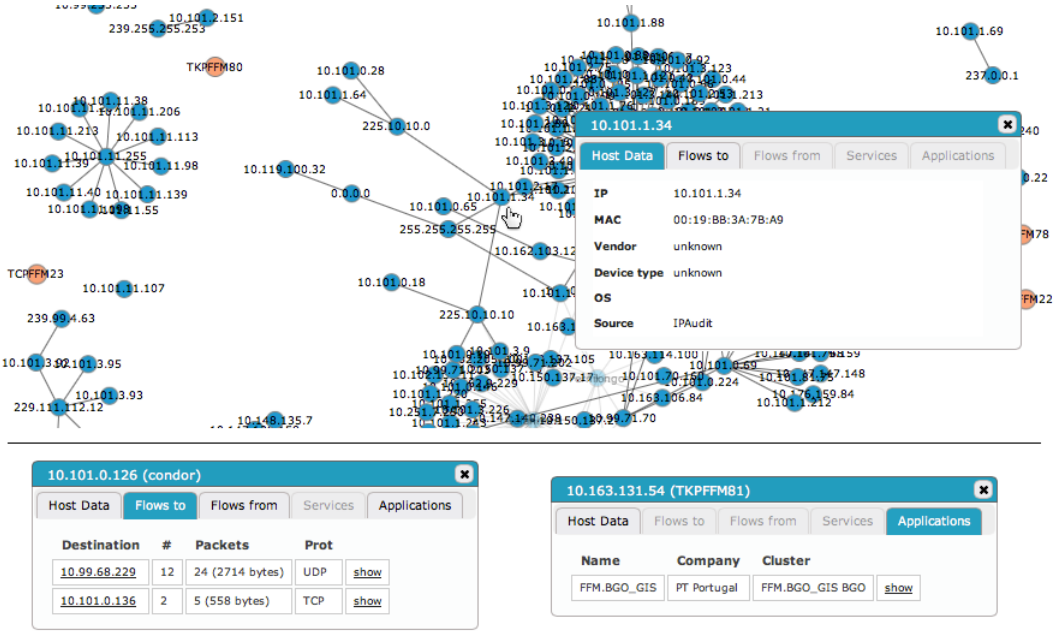

Figura 6.11: Informação detalhada de um ativo selecionado no grafo.

A figura 6.11, apresenta a implementação realizada por forma a mostrar os dados anteriormente referidos, note-se que a informação disponibilizada nos quatro últimos separadores encontra-se agrupada/resumida. Informação mais detalhada que poderá ser consultada através dos links disponíveis nos ditos agrupamentos.

No caso das listas de fluxos de comunicação, agrupados por protocolos e *hosts*, é possível aceder a cada agrupamento para consultar cada um dos fluxos detalhados individualmente (figura 6.12).

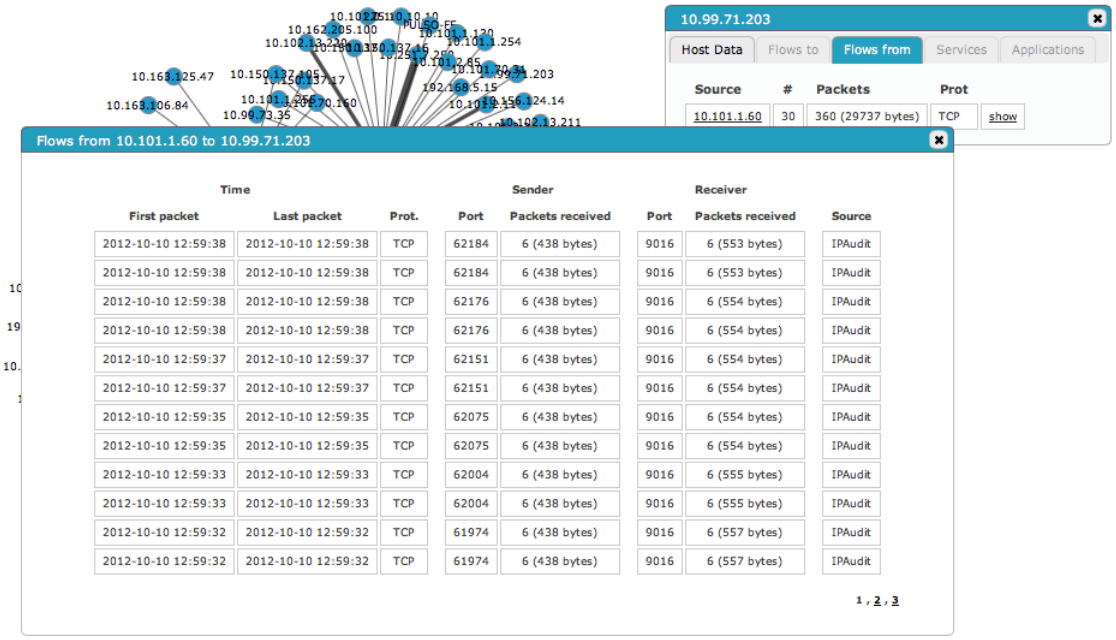

Figura 6.12: Informação detalhada dos fluxos de comunicação existentes entre dois *hosts* arbitrários, recorrendo ao protocolo TCP.

6.3.5 *Layout* Pulso/Discovery

O *layout* Pulso/Discovery toma como principal foco a representação dos ativos e da comunicação entre eles através de grafos, como já referido anteriormente. É baseado nesta ideia que o *layout* do portal foi desenhado.

Pretende-se que o foco principal do utilizador seja de facto a interação e visualização dos grafos de rede. É com base neste principio que se colocou o grafo exatamente no centro e constantemente presente como fundo da página. Ao mesmo tempo procurou-se que os restantes elementos que compõem o *layout*, interferissem o menos possível com a visualização do grafo.

Todos os menus e camadas de suporte são colocados preferencialmente em volta do grafo e quando não é possível, por uma questão de espaço, as camadas são colocadas de modo a que o utilizador possa controlar a sua posição dentro da área do grafo, isto é, todas as camadas adicionais têm opções para poderem ser movidas dentro do espaço disponível

da página, por forma a não obstruir a visualização do grafo. Além disso, existem opções que permitem ocultar estes elementos, ficando o grafo completamente visível.

Os menus de filtragem e de controlo de variáveis físicas do grafo são igualmente possíveis de fechar ou abrir conforme a necessidade do utilizador. Cria-se assim um ambiente em camadas/janelas dos elementos em torno do grafo.

A implementação das animações de entrada e saída dos elementos do canvas, assim como a interação com estes, recorreu à biblioteca jQuery. Foram também usados elementos jQueryUI para a construção de alguns dos elementos da página e *plugins* relacionados para construir determinados elementos mais complexos, nomeadamente o qTip[1] para algumas *tooltips*, e o multi-open-accordion[2] para a construção do menu de filtros.

O portal recorre à técnologia AJAX para a implementação de algumas das funcionalidades, permitindo que só partes da estrutura DOM da página sejam alteradas quando há a necessidade de comunicar com o servidor para gerar novos elementos e disponibilizar uma nova informação.

Alguns *screenshots* da ferramenta são apresentados nas figuras 6.13, 6.14 e 6.15, onde se pode verificar alguns pontos enunciados aqui.

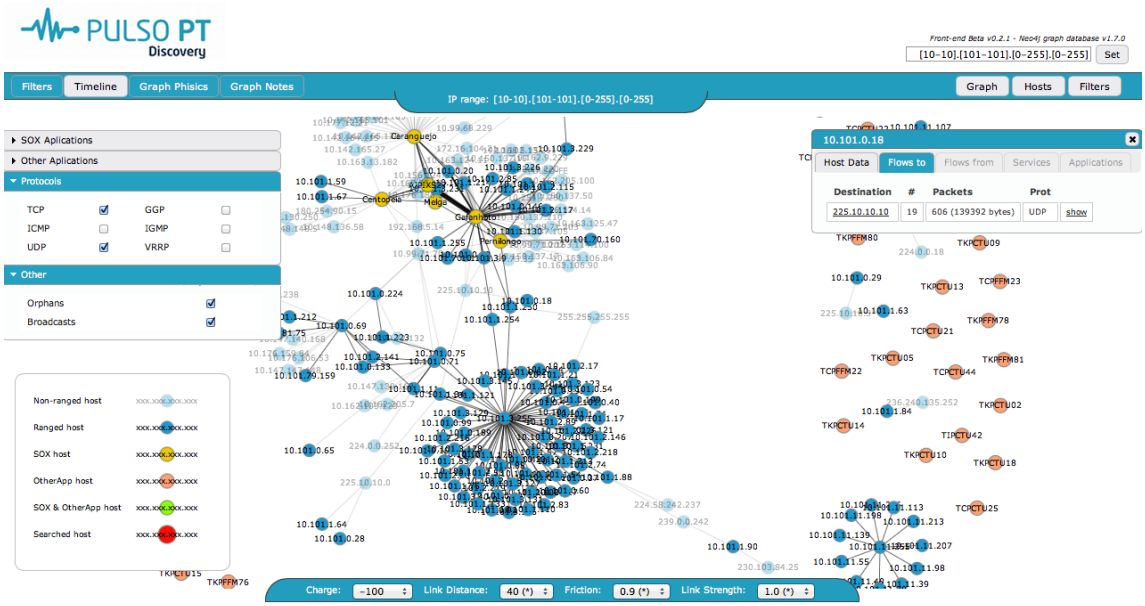

Figura 6.13: *Layout* da página Pulso/Discovery.

6.3.6 Interação com a base de dados

A interação com a base de dados no portal recorre à biblioteca Neo4j.rb[3] e consequentemente à Traverser API e ao Apache Lucene para a indexação de dados.

[1] `http://craigsworks.com/projects/qtip/`
[2] `http://code.google.com/p/jquery-multi-open-accordion/`
[3] `https://github.com/andreasronge/Neo4j`

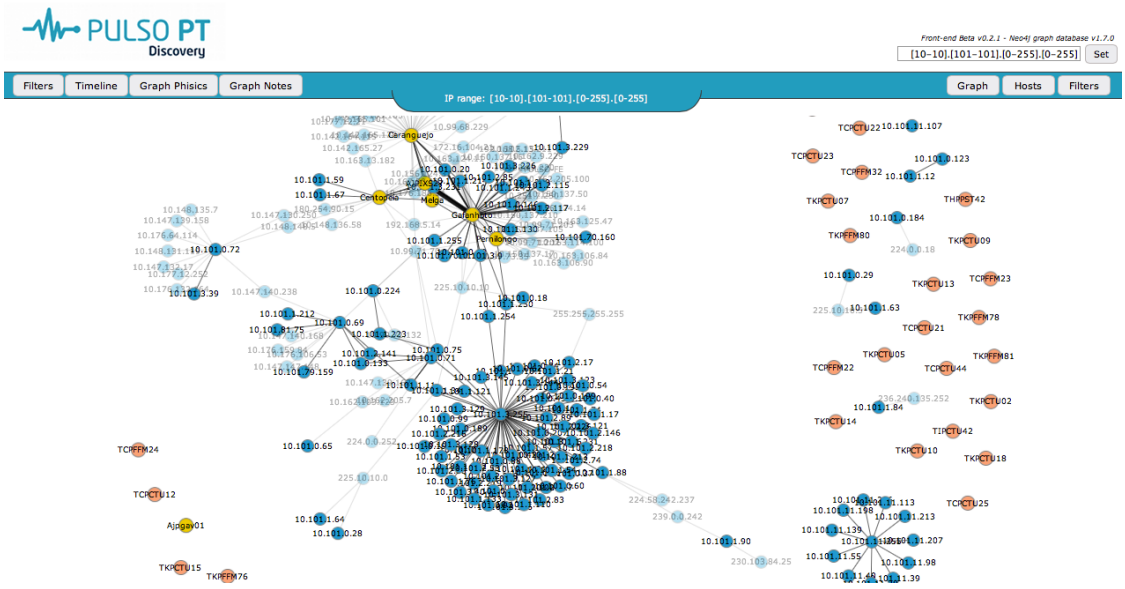

Figura 6.14: *Layout* da página Pulso/Discovery.

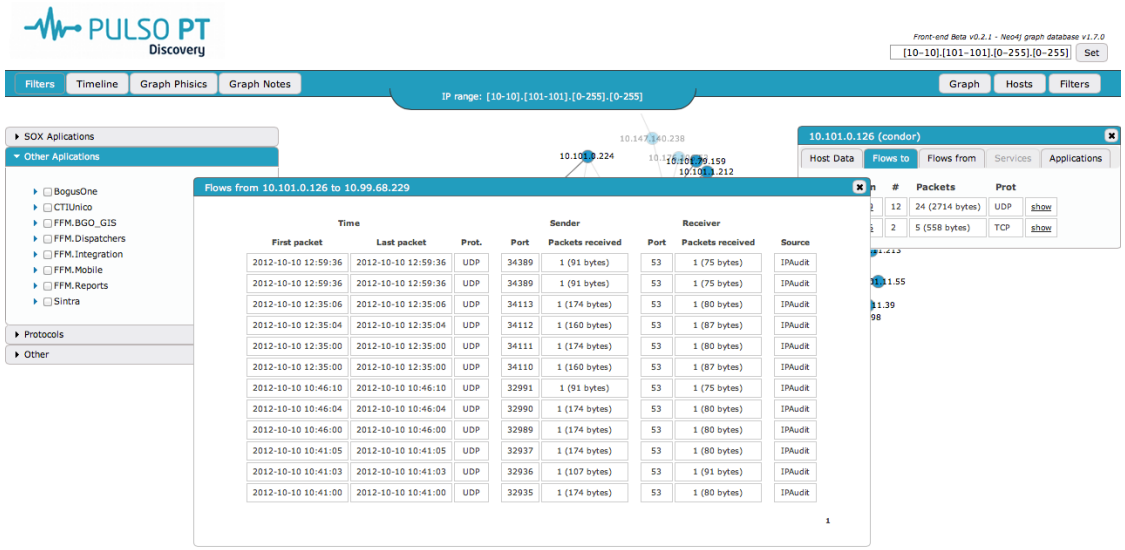

Figura 6.15: *Layout* da página Pulso/Discovery.

O portal Pulso/Discovery é totalmente implementado sobre a *layer* 3 do Neo4j.rb (secção 5.6), o que facilita muito a manutenção e organização de código, tirando partido das muitas das vantagens que o modelo MVC (*Model-View-Controller*) do Ruby on Rails oferece. Além disso, usar a *layer* 3 do Neo4j.rb, significa que a aplicação funciona muito bem com o Ruby on Rails versão 3 e com as restantes gemas disponibilizadas. Se no entanto, for necessário melhorar o desempenho em alguns pontos, será sempre possível recorrer às camadas mais baixas sem prejuízo de alterar a estrutura existente.

As consultas para a construção dos grafos são realizadas igualmente por combinação de visitas e indexação composta Lucene, dependendo do tipo de pesquisa e filtragem

realizados. Estes procedimentos são implementados em controladores Rails cuja resposta é gerada em JSON, servindo como *input* direto à biblioteca d3.js e ao módulo que constrói os grafos, facilitando assim a sua criação.

Os grafos são totalmente programados em Coffee Script[1] ao invés de puro JavaScript por uma questão de boa programação, uma vez que a ferramenta Ruby on Rails 3.1 usa esta linguagem por norma.

6.3.7 Bibliotecas utilizadas

Este módulo é desenvolvido em JRuby através da ferramenta Ruby on Rails v3.1 e faz uso das seguintes gemas (entre outras de menor relevância):

Neo4j.rb v2.0.0 Um *wrapper* JRuby para bases de dados Neo4j. Usa a biblioteca Neo4j para interação com a base de dados e para o uso da Traversal API e do Apache Lucene para consulta e indexação.

Neo4j-enterprise v.1.7.0 Para a ligação e interação da base de dados Neo4j.

Neo4j-will-paginate v.0.1.2 Para implementação de consultas paginadas à base de dados e respetiva interface pessoa-máquina, como por exemplo as listagens dos fluxos de comunicação entre ativos.

Json v1.6.6 Para o *parsing* e compilação de ficheiros e respostas JSON, nomeadamente para construção dos grafos em parceria com a biblioteca d3.js.

Coffee-rails v3.1.1 Adaptador JavaScript usado no Ruby on Rails, usado para programação dos grafos e de diversas funcionalidades do portal.

Jquery-rails v1.7.2 Biblioteca jQuery para o Ruby on Rails: usado para a simplificação de procura de elementos HTML, manipulação de eventos, animação e interação AJAX. Além desta biblioteca são usados dois *plugins* JQuery: QTip para construção de *tooltips* e MultiOpenAccordion para construção do menu de filtros.

Jquery-ui-rails v1.8.21 Pacote de temas e técnicas de interação pessoa-máquina para o Ruby on Rails utilizado para a contrução do *layout* do portal.

D3.js v2 Biblioteca JavaScript para a visualização dos grafos *force-directed* de rede.

[1] `http://coffeescript.org/`

Capítulo 7

Avaliação

O modelo teórico e o desenho apresentado no capítulo 5 (e que foram implementados em três componentes distintos, como descreve o capítulo 6) foram submetidos a vários testes e avaliações ao longo do seu desenvolvimento.

Neste capítulo são descritos em detalhe os procedimentos e o ambiente utilizado nos testes apresentados, avaliando e disponibilizando os resultados obtidos referentes às componentes de *back-end* (módulos DiscoveryZookeeper e DiscoveryServer) e *front-end* (módulo DiscoveryPortal).

7.1 Procedimentos gerais de avaliação utilizados

A avaliação experimental teve por base a captura passiva e ativa de tráfego por parte de sondas em diferentes interfaces de rede e em diferentes alturas do dia, com mais ou menos volumes de dados (tema de um outro projeto realizado paralelamente a este) consoante o tipo de testes que se pretendia efetuar (testes de desempenho/escalonamento, testes funcionais do servidor ou testes do portal). É importante realçar que uma das sondas utilizadas pelos testes, está localizada num *datacenter* da empresa, e recorre a uma placa de rede de alto desempenho para a captura de tráfego, capturando quantidades enormes de dados, desafiando a plataforma a diversos níveis.

Colocou-se o sistema sobre avaliação na rede empresarial durante várias semanas, onde foi possível que várias pessoas do departamento onde este projeto está inserido, acedessem ao portal *web* e testassem a interface pessoa-máquina, navegando sobre a informação derivada dos dados fornecidos pelas sondas de rede.

Várias melhorias foram realizadas com base nos comentários, sugestões e pareceres dos diferentes avaliadores, esta foi a forma mais eficaz de avaliar a ferramenta, nomeadamente em termos de desempenho e interação pessoa-máquina.

As diferentes versões foram sendo disponibilizadas na rede, à medida que se encontravam funcionais por forma a receber novos *feedbacks* e novas avaliações pelos membros responsáveis pelo projeto para que estas fossem consecutivamente melhoradas a cada

iteração.

Outra forma de avaliação consistiu na realização de reuniões de projeto onde se discutiam diferentes tópicos com vista a melhorar a aplicação.

7.2 Apresentação dos resultados

Foram realizados testes gerais à plataforma desenvolvida de modo a apresentar resultados concretos que avaliem e testem a ferramenta desenvolvida.

Seguidamente é apresentado o ambiente utilizado, as configurações utilizadas em cada um dos módulos desenvolvidos e os respectivos resultados dos testes realizados às componentes de *back-end* e *front-end*.

7.2.1 Ambiente utilizado

As três componentes deste projeto, DiscoveryServer, DiscoveryZookeeper e DiscoveryPortal foram instaladas numa máquina com 24 núcleos de 3.33GHz com uma memória total de 64GB num sistema Red Hat Enterprise Linux Server *release* 5.3 (Tikanga). Testou-se e avaliou-se os diferentes módulos do projeto nesta máquina, pois é a máquina onde irá funcionar futuramente o servidor do sistema Pulso/Discovery em ambiente de produção.

Foram instalados o Java v1.6, Ruby v1.9.2, JRuby v1.6.5, Neo4j v1.7 e o Ruby on Rails v3.1 com as respetivas gemas (listadas no secção 6.3.6). O portal *web* foi testado através de uma servidor *web* WEBrick[1].

Os testes realizados recorreram aos dados fornecidos pelas sondas Pulso/Discovery do projeto desenvolvido paralelamente a este, que utiliza uma placa de rede de alto desempenho e capturando em média por hora, 48 GB de tráfego o que corresponde a aproximadamente 220.000.000 pacotes (média calculada por dados medidos pela ferramenta IPTraf[2] durante 3 dias).

Quanto às mensagens provenientes das sondas, é enviada uma a cada 500.000 pacotes capturados por análise passiva e uma por cada gama de endereços IP analisado ativamente (recorrendo à ferramenta NMAP). As mensagens provenientes de análise ativa são enviadas das sondas para o servidor manualmente, sempre que se mostre necessário.

Os resultados apresentados de seguida referem-se a um espaço temporal de 1 hora, entre as 11h13 e as 12h13 do dia 24 de Setembro de 2012. A base de dados foi totalmente apagada antes dos testes, para se perceber melhor a sua evolução em termos de dados.

[1] `http://apidock.com/ruby/WEBrick`
[2] `http://iptraf.seul.org`

7.2.2 Configurações

De seguida são apresentados as configurações definidas para os diferentes módulos desenvolvidos a quando dos testes efectuados ao *back-end* (DiscoveryZookeeper e DiscoveryServer) e *front-end* (DiscoveryPortal). Os resultados serão apresentados respectivamente na secção 7.2.2 e 7.2.3.

DiscoveryZookeeper

Este módulo foi configurado para a criar o *cluster* Neo4j em modo de alta disponibilidade com três instâncias independentes e localizadas localmente os servidor, com um intervalo de tempo de sincronização de dados de $1ms$.

A instância *master* foi configurada para ser utilizada pelo módulo DiscoveryServer por razões de eficiência, uma vez que é nesta base de dados irão ser realizadas a maior parte das escritas e grande parte das consultas. Uma outra instância foi configurada para ser utilizada pelo módulo DiscoveryPortal.

DiscoveryServer

Os testes apresentados recorreram a um ficheiro de configuração idêntico ao apresentado na figura 6.5.

É definido um acesso à base de dados Neo4j localmente em modo de comunicação embebido, duas portas de comunicação distintas (uma para cada tipo de dados fornecido pelas sondas), um acesso remoto à base de dados Pulso, um ficheiro de configuração referente à instância Neo4j *master*, entre outros parâmetros referentes ao Active Model e a *logs* de sistema.

Note-se que o parâmetro de configuração que mais impacto tem sobre o desempenho nos testes realizados é o tipo de comunicação com a base de dados que pode ser REST ou embebido.

O modo REST foi devidamente testado em fazes preliminares do projeto, não oferecendo garantias suficientes em termos de desempenho quando posto à prova com grandes volumes de dados provenientes de diversas sondas Pulso/Discovery. Este modo de comunicação com a base de dados é mais lento quando comparado com o uso direto da API embebida, tendo um maior risco de se tornar um ponto de estrangulamento para o sistema, a quando de situações de captura extrema de dados. Assim, o modo REST foi um pouco deixado de parte após alguns testes, embora apresente algumas vantagens discutidas mais a frente na secção 8.1.

DiscoveryPortal

Este módulo é configurado para usar uma instância (réplica) do *cluster* de alta disponibilidade Neo4j. Instância que está localizada na mesma máquina onde se encontra

instalado este módulo.

7.2.3 *Back-end* - Servidor e bases de dados

Os dois módulos que compõem o *back-end* (DiscoveryZookeeper - secção 6.1 e Discover Server - secção 6.2) foram testados e avaliados com recurso a tempos de processamento e armazenamento.

Os valores utilizados para a construção dos diferentes gráficos e para o cálculo de médias e outras informações estatísticas tiveram origem nos *logs* produzidos pelo módulo DiscoveryServer desenvolvido (disponibilizados na tua totalidade em anexo na secção A.1). Estes valores foram tratados e organizados numa tabela (em anexo na secção A.1.1), que serviram para produzir os gráficos apresentados de seguida.

Os *logs* anteriormente referidos disponibilizam informações relativamente às mensagens recebidas pelo servidor e provenientes da sonda de rede que alimenta estes testes (referida na secção 7.2). São apresentados tempos totais de processamento e armazenamento para cada uma das mensagens, assim como as quantidades de nós, relações e propriedades que cada uma das mensagens proporcionou na base de dados. Os *logs* incluem também informações referentes à instância Apache Zookeeper utilizada por este módulo através da biblioteca Log4j.

Este módulo quando executado, realiza um conjunto de procedimentos indispenáveis antes de poder receber qualquer tipo de mensagens (como confirma os *logs* apresentados em anexo na secção A.1):

1. É lido o ficheiro de configuração do módulo para definir um conjunto configurações necessárias descritas na secção 7.2.2;

2. É criado uma ligação com a base de dados Neo4j em modo de alta disponibilidade (ligação à instancia Apache Zookeeper respetiva);

3. É criado, caso ainda não exista, uma estrutura adicional à base de dados para a organização de nós e relações para que o a biblioteca Neo4j.rb (secção 5.6) mapeie os dados diretamente em classes Ruby, tal como explicado na secção 6.2.3;

4. São realizadas duas consultas, uma à base de dados Pulso e outra a ficheiros referentes a dados de aplicações SOX por forma a obter informações atualizadas de aplicações, subsistemas e *clusters* existentes assim como os ativos que os compõem;

5. São lançadas duas *threads* para receber dados provenientes das sondas de rede Pulso/Discovery, uma para dados de análise passiva e outra para dados de análise ativa do tráfego de rede.

A partir deste ponto, o servidor está pronto a receber mensagens. Os dados inseridos na base de dados Neo4j deste módulo são automaticamente sincronizados com o repositório do módulo DiscoveryPortal, estando desde disponíveis de ser consultados.

Durante o tempo em que os testes se efetuaram, os dados que alimentaram o servidor em termos de metadados tiveram origem numa consulta à base de dados Pulso e outra consulta a ficheiros de metadados sobre aplicações SOX. No caso da informação proveniente das sondas, foram recebidas 4 mensagens pelo servidor ativo (uma referente à análise de ativos SOX, outra referente à classe de endereços 10.101.68.0/24 e a duas referentes a conjuntos de endereços diversos e conhecidos pela aplicação IP Audit) e 232 mensagens recebidas pelo servidor passivo (onde cada uma conta com aproximadamente 500.000 pacotes de rede capturados).

Em termos práticos, os dados recebidos traduziram-se na descoberta de 923 ativos de rede, 413.214 fluxos de comunicação (aproximadamente 116.000.000 pacotes de rede), 19 aplicações, 55 subsistemas e 87 *clusters*, caracterizados por um total de 4.134.587 propriedades/atributos.

Resultados obtidos

Na figura 7.1 são apresentadas as quantidades de relações e propriedades Neo4j inseridas ao longo do tempo, assim como a respetiva evolução da quantidade destes elementos na base de dados. Os picos do gráfico são referentes à receção de novas mensagens, umas com mais conteúdo e informação do que outras. Aqui há uma certa correspondência nas duas quantidades inseridas, uma vez que refletem novos fluxos de comunicação e as respetivas propriedades

Na figura 7.2 são apresentadas as quantidades de nós Neo4j inseridos ao longo do tempo, assim como a respetiva evolução da quantidade destes elementos na base de dados. É de notar uma certa quebra em termos inserção de nós na base de dados ao longo do tempo uma vês que estes refletem, na grande maioria, ativos de rede. Ativos que se tornam conhecidos logo nas primeiras mensagens recebidas com dados provenientes da análise ativa e através dos primeiros fluxos de rede recolhidos por análise passiva. Posteriormente são detetados apenas os ativos que realizem algum tipo de comunicação, e que por estarem inativos aquando da análise ativa, não foram descobertos. Outro tipo de informação modelada através de nós são as aplicações, os subsistemas, os *clusters*, e os serviços detetados e que tal como os ativos, têm tendência a estabilizar ao longo do tempo embora existam sempre mudanças.

Na figura 7.3 é apresentado um gráfico de dispersão de tempos de processamento e armazenamento de cada uma das mensagens recebidas pelo servidor quando comparado ao número de nós, relações e propriedades (informação) Neo4j resultantes. É de notar que há uma certa linearidade nos tempos de processamento e armazenamento dependendo do volume de informação em cada mensagem. As mensagens recebidas pelo servidor ativo

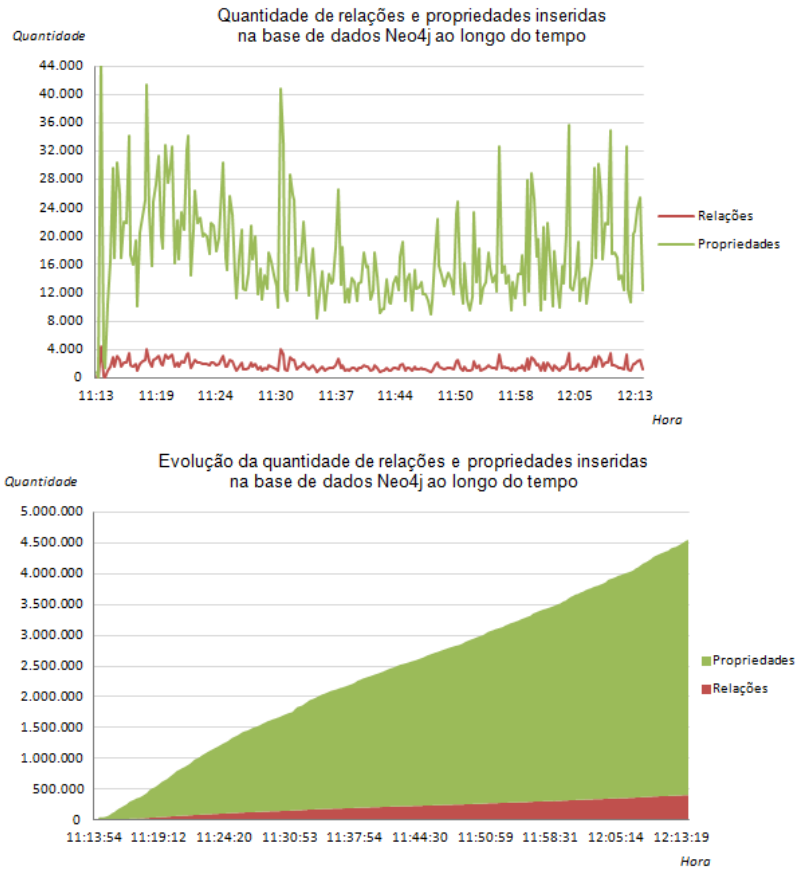

Figura 7.1: Gráficos referentes a quantidades de relações e propriedades Neo4j inseridas ao longo do tempo, assim como a respetiva evolução da quantidade destes elementos na base de dados.

geralmente provocam tempos mais altos de processamento, pois traduzem-se numa maior quantidade de informação a ser processada e num maior nível de estruturação ao nível do modelo de dados.

Ainda sobre tempos de processamento, note-se que durante o espaço temporal (pouco mais de uma hora) utilizado, precisou de 948,578 segundos (aproximadamente 15,8 minutos) para processar, tratar e armazenar os dados de acordo com a modelação apresentada (secção 6.2.3), o que é bastante bom tendo em consideração as quantidades de dados capturados pela sonda de alto desempenho em termos de fluxos de comunicacão.

7.2.4 *Front-end* - Portal Pulso/Discovery

Este módulo foi testado e avaliado através das funcionalidades e visualizações oferecidas pela aplicação *front-end* desenvolvida em Ruby on Rails, com um principal foco nos grafos de rede gerados e pelos respetivos tempos de processamento de cada um. Adicionalmente são apresentados outros resultados de interação com a ferramenta para além das funcionalidades e exemplos já apresentadas na secção 6.3.

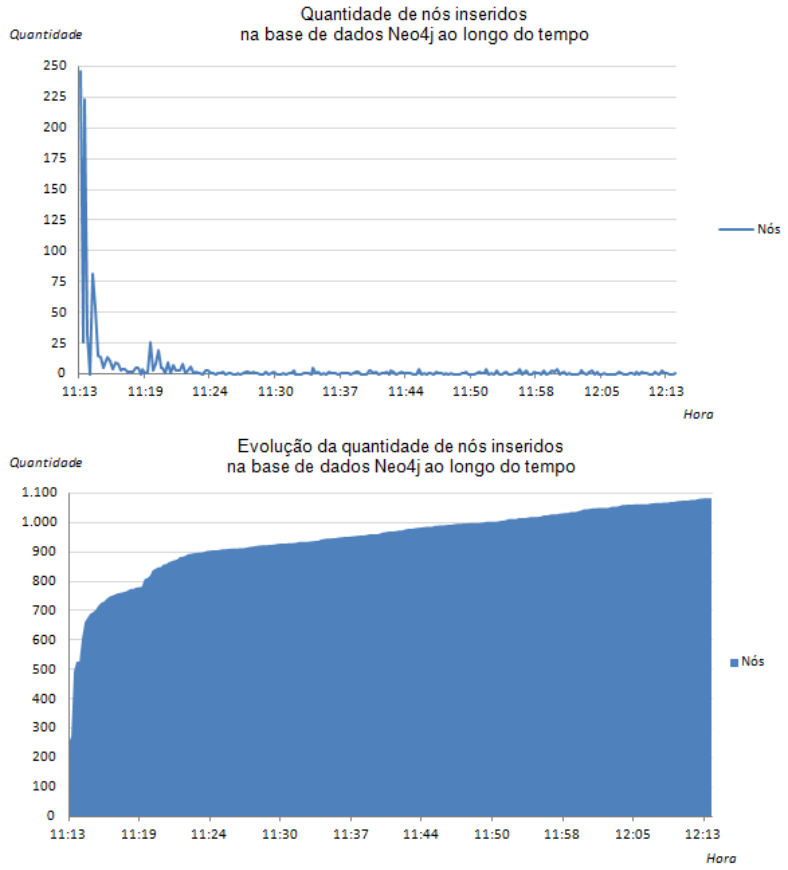

Figura 7.2: Gráficos referentes a quantidades de nós Neo4j inseridos ao longo do tempo, assim como a respetiva evolução da quantidade destes elementos na base de dados.

A informação e os resultados apresentados de seguida têm como base os dados armazenados pelo módulo *back-end*, referidos na secção 7.2.2. Os tempos de execução e dados estatísticos sobre o conteúdo dos diferentes grafos foram retirados diretamente dos *logs* Ruby on Rails do módulo DiscoveryPortal (disponibilizados em parte em anexo na secção A.2). Os *logs* apresentados disponibilizam informação relativamente aos pedidos HTTP realizados ao servidor para a construção dos grafos de rede pelo portal.

Resultados obtidos

As tabelas 7.1, 7.2 e 7.3 apresentam os dados referentes aos 9 grafos de rede construídos e disponibilizados pela aplicação durante o tempo de execução dos testes. As tabelas apresentam respetivamente, dados relativamente a tempos de processamento, quantidade de ativos representados e quantidades de ligações representadas.

Como pode ser observado, foram realizados três tipos de pesquisas distintas, uma à gama de endereços de todas as aplicações SOX, uma à gama de endereços interno 10.101.0.0/16 e outra ao endereço 10.101.1.60. Cada uma das configurações de pesquisa apresentados foram utilizados três vezes em diferentes alturas para perceber a evolução

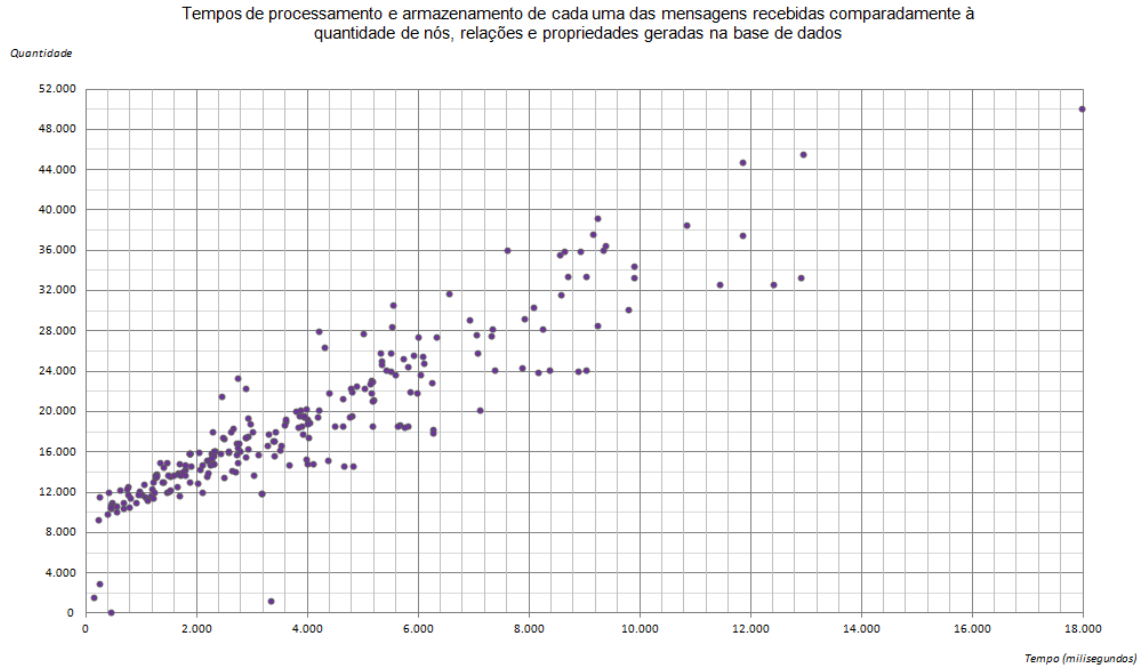

Figura 7.3: Gráfico de dispersão de tempos de processamento e armazenamento de cada uma das mensagens recebidas pelo servidor quando comparado com número de nós, relações e propriedades Neo4j resultantes.

dos grafos ao longo do tempo.

As tabelas apresentadas em 7.1, 7.2 e 7.3 mostram a informação relativamente aos 9 grafos gerados e apresentados nas figuras 7.4 (grafos G1 a G3), 7.5 (grafos G4 a G6) e 7.6 (grafos G7 a G9), para cada um dos grafos a tabela indica o número de ativos aplicacionais encontrados, o número de ativos da gama de endereços pesquisada encontrados, o número de ativos que embora se encontrem fora da gama de endereços pesquisada, comunicam diretamente os pretendidos, o total de ligações e fluxos correspondentes e o tempo de processamento para a visualização de cada um.

Os grafos refletem os fluxos de comunicação de todos os protocolos de rede existentes entre os ativos de rede procurados e entre ativos que, embora não pertenção à gama de endereços pretendida, apresentam fluxos diretos com os ativos procurados. São ignorados todos os ativos que embora pertençam à gama de endereços procurados, não apresentem qualquer evidência de tráfego. Estas e outras opções podem ser definidas através dos filtros desenvolvidos como é apresentado na secção 6.3.1.

As restantes funcionalidades do portal estão apresentadas segundo alguns exemplos na secção 6.3. Referente a estes dados utilizados pelos testes, são apresentadas informações referentes a um ativo arbitrário. Na figura 7.7 é apresentado os atributos do ativo, os serviços detetados e as aplicações a que pertence, sendo permitindo aceder por interacção *web* às ligações "*show*" respetivas para consultar mais informação. Na figura 7.8 é apresentado informação sobre fluxos com origem e com destino neste ativo, assim como uma

	Gama de endereços pesquisada	Hora	Tempo de processamento (ms)
G1	Aplicações SOX	11:22:29	1.763
G2	Aplicações SOX	11:38:20	1.648
G3	Aplicações SOX	11:53:44	1.479
G4	10.101.0.0/16	11:23:51	1.330
G5	10.101.0.0/16	11:42:09	1.260
G6	10.101.0.0/16	11:57:58	1.225
G7	10.101.1.60	11:24:28	1.330
G8	10.101.1.60	11:44:45	1.260
G9	10.101.1.60	12:04:05	1.225

Tabela 7.1: Dados relativos a cada um dos 9 grafos gerados pelo portal, nomeadamente gama de endereços pesquisada, a hora em que em foi gerado e o tempo de processamento necessário.

	Total de ativos		
	Aplicacionais	**Na gama de endereços**	**Fora da gama de endereços**
G1	7	7	46
G2	8	8	167
G3	8	8	251
G4	-	38	26
G5	-	225	107
G6	-	230	147
G7	-	1	52
G8	-	1	67
G9	-	1	72

Tabela 7.2: Dados relativos a cada um dos 9 grafos gerados pelo portal, nomeadamente número de ativos de rede encontrados e desenhados pelo grafo.

listagem dos fluxos entre este ativo e um outro também arbitrário.

7.3 Discussão sobre os testes efetuados

O *Back-end* correspondeu muito bem aos volumes de dados exigidos, apresentando tempos de processamento bastante bons (considerando o grande volume de pacotes capturados por análise passiva), sendo que uma futura integração com um maior número de sondas de rede com placas de rede ditas normais, ou mesmo de alto desempenho, não se será um problema.

Em desenvolvimentos futuros, seria uma vantagem agregar fluxos de comunicação em espaços temporais distintos e limitados através de abstrações ao nível da base de dados Neo4j (tal como referido e explicado na secção 8.3) de modo a facilitar as consultas e melhorar os tempos de processamento para este tipo de dados.

	Total de ligações	**Total de fluxos**
G1	66	1.005
G2	245	63.637
G3	352	115.065
G4	79	1.261
G5	511	22.029
G6	575	36.609
G7	52	6.274
G8	67	67.381
G9	72	104.005

Tabela 7.3: Dados relativos a cada um dos 9 grafos gerados pelo portal, nomeadamente o número de ligações desenhadas e número de fluxos abstraídos no grafo.

O *Front-end* correspondeu igualmente bem nos diferentes níveis de interação com os grafos de rede consultados, com tempos de espera bastante bons dado a riqueza dos dados disponíveis.

É de notar que representações de grafos de rede referentes a espaços temporais superiores a uma/duas horas (quando sujeitos a volumes de fluxos de comunicação capturados na ordem dos 116.000.000 pacotes, como estes testes demonstraram), tendem para representações demasiado complexas e confusas, com os respetivos tempos de construção a tornarem-se também demasiado longos.

Existem diferentes formas de lidar com este problema de complexidade dos grafos, uma é recorrer aos filtros desenvolvidos para a aplicação, limitando a informação disponível em cada momento. Caso futuramente se pretenda ir mais além, poderá recorrer-se à agregação de ativos em nós abstratos segundo determinadas propriedades mas para isto será necessário disponibilizar ao servidor Pulso/Discovery outro tipo de dados mais ricos.

Muitos dos testes essenciais ao módulo *front-end* têm foco na interação pessoa-máquina com os grafos de rede, filtros e janelas que compõem o portal *web* desenvolvido. A aplicação foi devidamente testada e avaliada por um conjunto de pessoas do departamento EDS-SI/TI onde se insere este projeto, obtendo-se apreciações muito construtivas que permitiram melhorar a aplicação. No entanto, tais testes/avaliações de interação pessoa-máquina são difíceis de apresentar neste relatório.

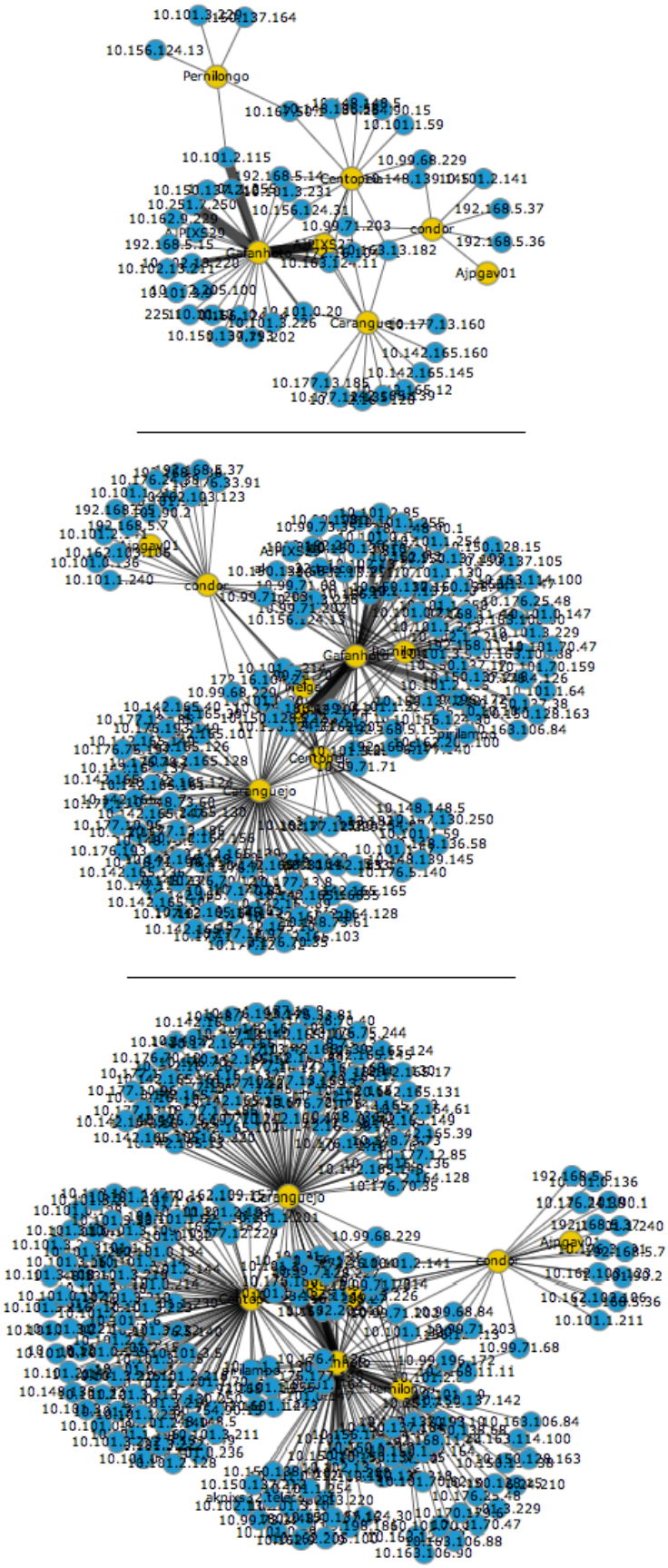

Figura 7.4: Grafos de rede gerados pelo módulo DiscoveryPortal sobre a gama de endereços de aplicações SOX em diferentes momentos durante a execução dos testes - Grafos G1, G2 e G3 respetivamente.

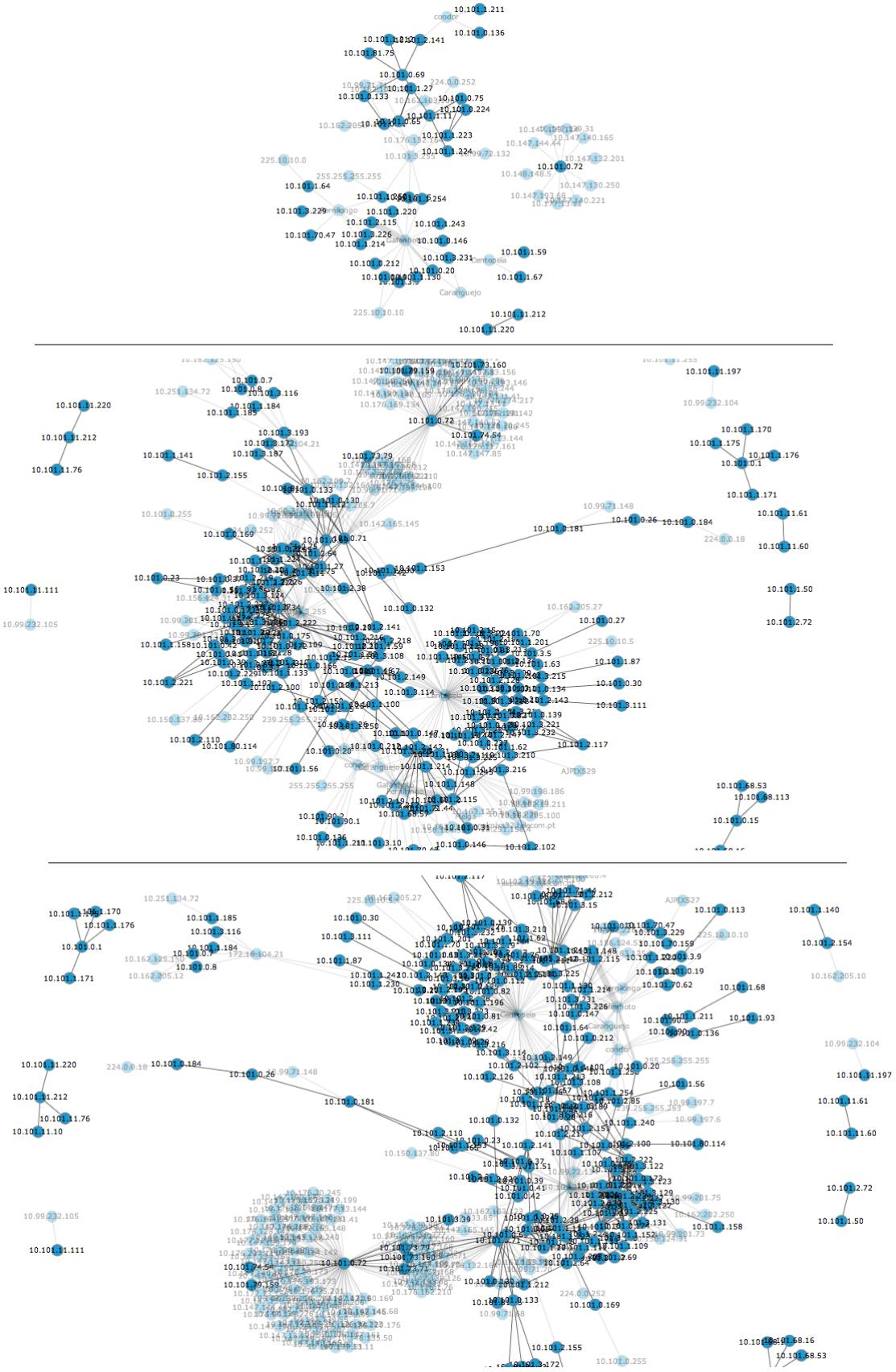

Figura 7.5: Grafos de rede gerados pelo módulo DiscoveryPortal sobre a gama de endereços 10.101.0.0/16 em diferentes momentos durante a execução dos testes - Grafos G4, G5 e G6 respetivamente.

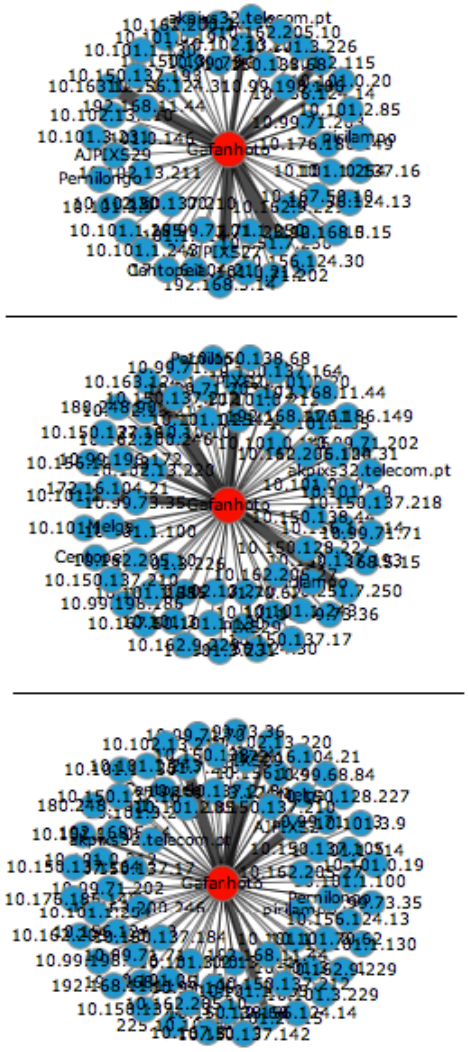

Figura 7.6: Grafos de rede gerados pelo módulo DiscoveryPortal ao IP 10.101.1.60 em diferentes momentos durante a execução dos testes - Grafos G4, G5 e G6 respetivamente.

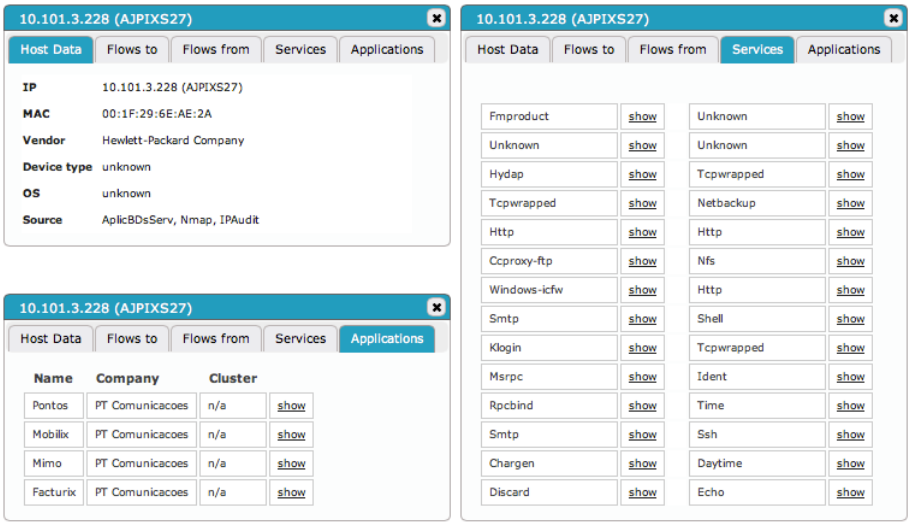

Figura 7.7: Interface pessoa-máquina que apresenta os atributos do ativo com o endereço 10.101.3.225, os serviços detetados e as aplicações a que pertence.

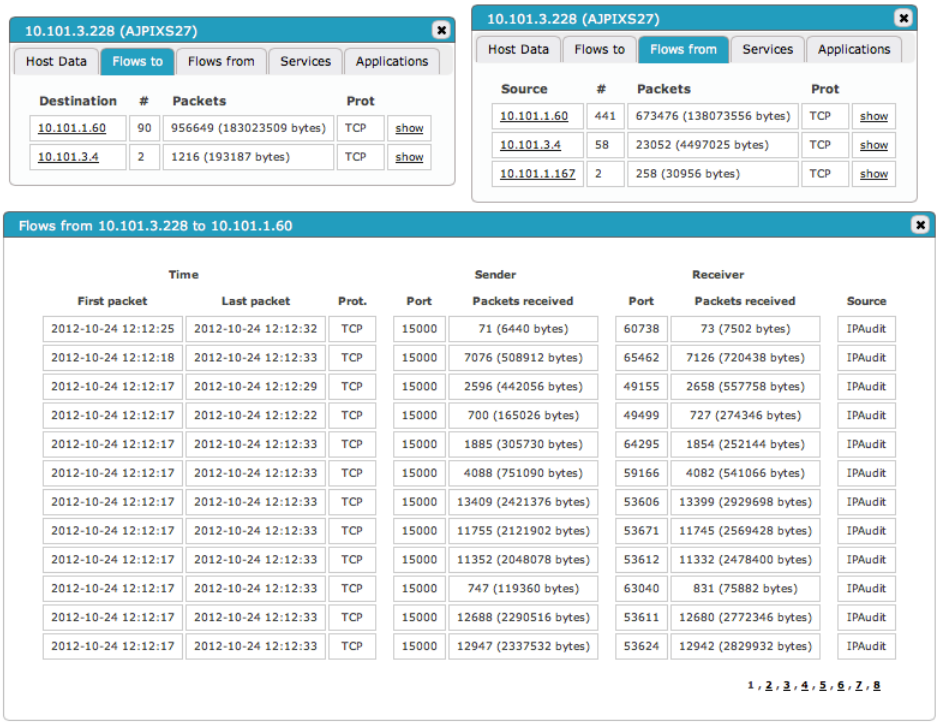

Figura 7.8: Interface pessoa-máquina que apresenta informação sobre fluxos com origem e destino no ativo ativo com o endereço 10.101.3.225, assim como uma listagem dos fluxos com origem neste ativo e como destino o endereço 10.101.1.60.

Capítulo 8

Discussão e trabalho futuro

Neste capítulo são discutidas algumas alternativas/opções de implementação realizadas e serão esclarecidos alguns tópicos importantes referentes a assuntos chave do projeto.

São discutidas algumas possibilidades de trabalho futuro, enumerando as mais valias que irão proporciona á plataforma e descrevendo possíveis abordagens de implementação.

Por fim, é apresentado uma síntese das dificuldades enfrentadas ao longo do desenvolvimento deste projeto e analisado o trabalho desenvolvido em comparação com o plano de trabalhos inicialmente ponderado, assim como as conclusões finais do projeto.

8.1 Alternativas gráficas para representação dos grafos

Foram estudadas diferentes ferramentas e bibliotecas *front-end* para responder ao desafio do projeto em disponibilizar informação de rede de uma forma interativa e útil. Entre as alternativas mais válidas encontra-se a biblioteca Arbor.js.

O Arbor.js é uma biblioteca especializada unicamente em grafos (ao contrário da biblioteca D3.js) construída com *web workers*[1] e JQuery[2]. Ao invés de tentar ser uma biblioteca que englobe muitas funcionalidades gráficas, esta procura especializar-se num algoritmo eficiente *force-directed*. Grande parte do controlo de desenho do *layout* do grafo é dado ao utilizador através de elementos SVG e HTML.

Esta, recorre-se igualmente ao algoritmo *Barnes-Hut n-body implementation* [3]. Tal como acontece com a biblioteca d3.js para a computação do *layout force-directed* dos grafos embora com uma implementação diferente da maioria das bibliotecas estudadas. Este algoritmo de adaptação dos nós e ligações do grafo é percetivelmente diferente. Tem como vantagem um melhor aproveitamento dos espaços vazios e consecutivamente uma menor taxa de oclusão de nós e arestas; como desvantagem um tempo maior de espera o que se torna bastante problemático em grafos demasiado complexos e quando a espaço de desenho dos grafos é limitado. O aspecto físico e matemático que gera o gráfico é

[1] `http://dev.w3.org/html5/workers/`
[2] `http://jquery.com/`

controlado totalmente pela ferramenta.

O design é influenciado por uma biblioteca existente, denominada *Traer Physics*[1] usada numa biblioteca denominada Processing[2], também ela ponderada a quando do desenho de uma solução para este projeto.

8.2 Neo4j REST vs Embedded e modo de alta disponibilidade

Foram estudadas diferentes soluções arquiteturais de integração da base de dados Neo4j entre o servidor e o portal *web*. Entre as diferentes opções foi testado o modo de comunicação que recorre a uma tecnologia REST para comunicação com a base de dados Neo4j, o que facilitaria em muito a integração com a ferramenta Ruby on Rails e a respectiva biblioteca Neo4j.rb, já referida anteriormente na secção 6.3.5.

Acontece que tal tecnologia, aplicada à necessidade que o sistema Pulso/Discovery tem em termos de armazenamento de grandes volumes de dados poderia tornar-se futuramente um ponto de estrangulamento para o sistema em ambientes de captura de dados extrema. Por forma a evitar este potencial cenário, optou-se pelo modo embebido, onde se comunica com a base de dados Neo4j de uma forma mais direta, recorrendo à API da base de dados. Este modo embebido, de facto trouxe melhores resultados, oferecendo melhores garantias, mas criando um problema com a biblioteca Neo4j.rb utilizada no módulo DiscoveryPortal.

Segundo Andreas Ronge[3], um dos responsáveis pelo desenvolvimento da biblioteca Neo4j.rb, não deve existir mais do que um processo a aceder à base de dados Neo4j, nomeadamente através da biblioteca Neo4j.rb, o que induziu mais uma vantagem em usar o *cluster* de base de dados em modo de alta disponibilidade e disponibilizar assim ao DiscoveryPortal, uma base de dados dedicada.

Futuramente, podem existir vantagens em optar-se pelo modo REST. A REST API disponibilizada pela distribuição usa HTTP e JSON, tecnologias de fácil integração que podem ser usadas a partir de diversas linguagens e plataformas. Esta API é projetada para a descoberta pois pode ser iniciada com um GET ao *service root* e a partir daqui descobrir outras URI's para realizar outras requisições à base de dados. Com isto, temos um servidor onde múltiplas aplicações poderão usufruir da base de dados, acedendo diretamente ou através de um *web browser* o que poderá tornar-se interessante se o foco do projeto for outro e caso se pretenda uma abertura diferente aos dados Pulso/Discovery.

[1] `http://murderandcreate.com/physics/`
[2] `http://www.processing.org/`
[3] `http://twitter.com/ronge`

8.3 Trabalho futuro

Durante o processo de desenho e desenvolvimento deste projeto, que recorreu a uma metodologia baseada num processo refletivo e progressivo de resolução de problemas como descrito no secção 1.3, foram discutidas e ponderadas diversas ideias para ferramenta, quer por mim, quer pela equipa de trabalho responsável pelo projeto. Na verdade foram ponderadas e discutidas dezenas de ideias e funcionalidades, onde apenas algumas de facto acabaram por ser implementadas, aquelas que se mostraram mais importantes e prioritárias, tendo em consideração os dados a que a ferramenta teve acesso em determinados momentos e segundo os requisitos funcionais impostos.

Esta secção apresenta e discute algumas funcionalidades e melhorias importantes a desenvolver futuramente.

8.3.1 Eficiência na procura de ativos e relações

A pesquisa por ativos ou relações é um assunto chave para este projeto e que pode ser melhorado futuramente em alguns aspetos.

No caso dos ativos a melhoria poderá passar pela inclusão de novos dados que os agrupem segundo propriedades úteis tais como localização geográfica, localização física dentro de departamentos, segundo as funções que desempenham na rede, entre outras que se mostrem relevantes, num modo idêntico ao que foi feito com ativos de aplicações. Estes dados poderão ser consultados de fontes como a CMDB ou recorrendo ao sistema IPAM (IP Address Planning and Management System) existente na PT Comunicações que disponibiliza informação relevante sobre os diferentes *hosts* conhecidos, entre outras ferramentas a ponderar.

A possibilidade de visualizar grupos de ativos com propriedades semelhantes torna as visualizações mais interessantes, tirando assim um maior partido das funcionalidades da biblioteca gráfica que permite agrupar nós semelhantes num nó genérico, o que faz com que o grafo transmita uma melhor noção de profundidade em colaboração com as atuais noções de transparência e tamanho atribuídos a nós fora da gama de endereços procurados.

A visualização de grupos de ativos (através da agregação segundo propriedades que tenham em comum) permite observar relações de comunicação entre grupos como um todo. Isto possibilita ao utilizador consultar os fluxos de comunicação genéricos de todos os elementos que constituem o grupo. Na prática, a visualização iria disponibilizar um evento de interação sobre o nó que representa a abstração, permitindo-se depois encapsular/expandir os grupos de ativos. É apresentada uma imagem de um grafo com estas características na figura 8.1, onde o tamanho dos nós genéricos (grupos de ativos) refletem a quantidade de ativos que os constituem.

Cada um dos nós genéricos pode representar uma propriedade de rede útil, abstraindo

Figura 8.1: Exemplo de abstração de propriedades arbitrárias em nós genéricos através da biblioteca d3.js. Neste exemplo os nós a laranja e a verde (escuro), estão expandidos.

um conjunto de ativos com essa mesma propriedade. Esta é uma ideia para melhorar a visualização dos grafos de rede, que dado a escassez de metadados lógicos que os agrupem (a não ser relativamente a aplicações) acabou por não se realizar, visto não se mostrar útil sobre nenhum dos atributos dos dados que a ferramenta atualmente tem acesso.

No caso das aplicações, e especificamente nas aplicações monitorizadas pelo sistema Pulso os agrupamentos de nós mostraram-se pouco úteis uma vez que não foram capturadas quaisquer evidências de tráfego entre as diferentes aplicações o que se reflete num grafo genérico totalmente desconexo e sem propriedades úteis a analisar.

As diferentes aplicações SOX utilizam muitas vezes ativos comuns para o seu funcionamento, tornando este segundo nível confuso e suscetível de interpretações erradas já que um nó pertence geralmente a mais do que um nó genérico. Esta ideia foi posta de parte (para este tipo de dados) pois ter-se-ia que recorrer a mais do que um nó para a representação do mesmo ativo, tornando-se totalmente incoerente com o que é expectável por parte do utilizador.

Esta é uma funcionalidade muito interessante para futuros desenvolvimentos da plataforma, nomeadamente sobre dados que permitam agrupar ativos de rede de uma forma útil.

8.3.2 Linha temporal de fluxos de comunicação

Sobre as relações de comunicação, será interessante procurar fluxos específicos em intervalos de tempo limitados e bem definidos. É certo que esta informação é disponibilizada ao utilizador, através da ordenação dos fluxos pelos seus *timestamps*, mas seria

muito mais útil refletir apenas dados referentes a um intervalo de tempo específico no grafo, filtrando assim essa informação. Não tendo sido dada prioridade à implementação desta funcionalidade no decorrer do projeto, fica aqui a nota para um possível trabalho futuro.

Há diferentes formas de implementação desta funcionalidade. A mais simples e rápida seria recorrer a uma indexação simples dos fluxos, indicando o espaço temporal a que cada um pertence, por exemplo em intervalos de tempo de uma hora.

Outra forma de resolver este problema, aí sim tirando o máximo partido da base de dados Neo4j, seria através da criação de uma estrutura de nós e relações adicionais que organizassem de facto os fluxos em espaços temporais na base de dados, limitando o número de nós visitados nas respetivas consultas.

Mais especificamente, a implementação do modelo de dados poderá recorrer à criação de nós genéricos que representem uma abstração de um espaço temporal limitado, bastando depois relacionar a estes nós abstratos, os fluxos de comunicação realizados dentro do espaç temporal respetivo. As consultas através de visitas, apenas teriam que recorrer a estes nós abstratos para obter todos os fluxos para um dado intervalo de tempo e cruzando estes dados com uma outra visita já implementada que extrai apenas os fluxos dos ativos pretendidos. Com esta opção a pesquisa por nós e relações de um dado espaço temporal, em conjunto com a indexação Apache Lucene, torna-se mais eficiente.

Com a indexação e organização dos dados em espaços temporais implementados na base de dados, bastaria depois implementar a respetiva opção de filtragem ao portal *web*, em que nas visualizações dos grafos de rede, apenas se incluíam os fluxos que pertenciam aos intervalos selecionados, ignorando todos os outros.

Parte desta funcionalidade está já implementada (figura 8.2), nomeadamente no portal *web*, com filtros específicos para lidar com o que foi descrito, faltando apenas juntar ao *back-end* a nova visita correspondente a este filtro.

8.3.3 Comparação de grafos

Outra das funcionalidades ponderadas foi a comparação de grafos de comunicação capturados em diferentes espaços temporais por forma a detetar eventos fora do normal funcionamento da rede o que poderá enunciar um problema de segurança importante.

Através da comparação de grafos poderá deduzir-se novas propriedades ou eventos de rede e fornecer esses resultados ao utilizador que terá então a capacidade de perceber o porquê da existência de tais eventos e se constituiem ou não um problema.

Poder-se-á por exemplo, comparar grafos de rede relativamente a conjuntos de ativos específicos (pertencentes a determinadas gamas de endereços IP ou a aplicações) em espaços temporais delimitados diferentes e assim perceber padrões de comunicação que se desviam do normal funcionamento da rede (quando comparado com outros grafos em espaços temporais idênticos).

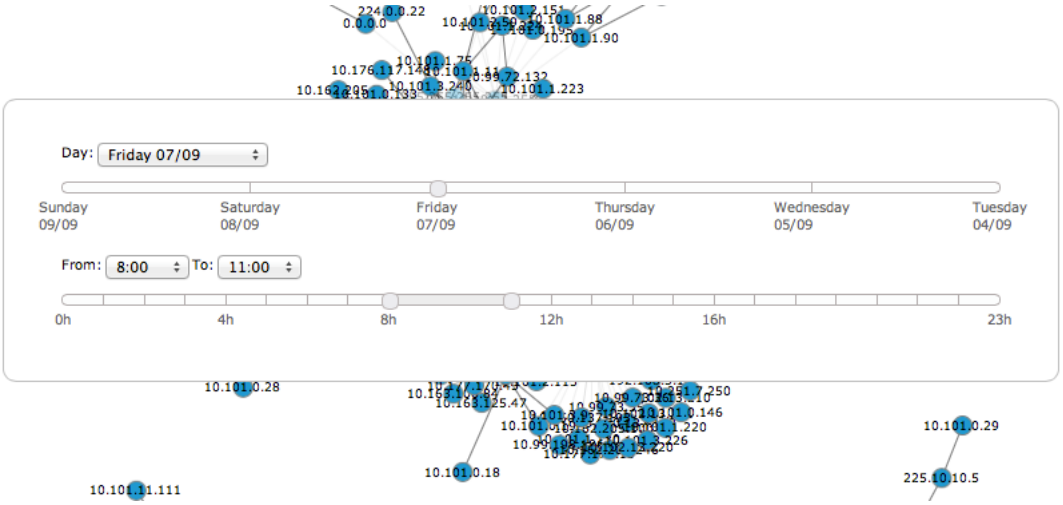

Figura 8.2: Implementação da interface pessoa máquina que possibilita informar o *back-end* do portal Pulso/Discovery do intervalo temporal que o utilizador pretende filtrar.

Para isto, poder-se-á recorrer a diferentes abordagens, dependendo da complexidade dos grafos envolvidos. Os algoritmos de grafos disponibilizados pela implementação Neo4j poderão ser uma solução, assim como a agregação de dados e metadados conhecidos, através de abstrações que mais eficazmente se permita comparar grafos distintos.

Ao nível da interface pessoa-máquina, é necessário desenvolver mecanismos para mostrar tais propriedades através do grafo de rede, permitindo ao analista perceber onde pode ou não existir tais eventos que necessitam de ser analisados em mais pormenor. Estes mecanismos podem recorrer a atributos visuais que permitam realçar determinado evento detetado segundo este mecanismo de comparação.

8.3.4 Visualização, interação e novos tipos de dados

A interação, como já descrito anteriormente, é baseada na visualização de grafos de rede e na navegação de informação sobre ativos e fluxos de comunicação. Pode tornar-se importante no futuro melhorar este modo de interação.

Assim que se recorrer a ferramentas que deduzam os protocolos dos fluxos de comunicação ao nível aplicacional, será possível deduzir problemas de segurança mais úteis, uma vez que grande parte dos problemas, hoje em dia, tendem a surgir a esse nível. Com estes dados é possível construir filtros mais eficazes e perceber que tipo de protocolos aplicacionais são utilizados na rede empresarial PT e entre que dispositivos. Por exemplo, certos subsistemas responsáveis por *front-ends*, bases de dados, entre outros, é apenas expectável que comuniquem recorrendo a um conjunto protocolos específicos, por questões de segurança.

Além das técnicas de filtragem, existem outras possibilidades de trabalho futuro relacionados com a interação com os grafos de rede. Uma das possibilidades passa por

implementar uma funcionalidade que a biblioteca d3.js oferece, a técnica *fisheye distortion*[1] para os grafos de rede para ajudar à visualização e compreensão de grafos de rede mais abrangentes. A complexidade visual de alguns grafos gerados pela plataforma Pulso/Discovery pode em parte ser resolvida através desta técnica que foi lançada muito recentemente, não tendo sido por isso ponderada a sua implementação atempadamente.

Além desta técnica, a biblioteca d3.js oferece uma vasta gama de visualizações ajustadas a diferentes tipos de informação, que se mostram altamente personalizáveis e que poderão tornar bastante úteis paralelamente aos grafos de rede.

Sobre a personalização do grafo através de formas, cor, níveis de transparência, tamanho, entre outros, é interessante estudar até que ponto abstrair certo tipo de informação no grafo é útil para que o utilizador tenha uma maior perceção da informação nele contida.

Isto pode ser realizado estudando as várias formas de representação SVG[2] existentes para representação dos ativos consoante uma dada propriedade (como o tipo de dispositivo), ou através de outros atributos gráficos que de facto ajudem à compreensão da informação, melhorando a perceção dos grafos. É possível também recorrer a imagens embora esta opção, no meu ponto de vista, adicione demasiada complexidade aos grafos, sendo preferível distinguir elementos recorrendo à cor e a diferentes formas geométricas/símbolos.

No caso das ligações entre nós do grafo, podem usar cor ou mesmo diferentes formas geométricas como por exemplo setas (figura 8.3), ao invés de linhas, e assim adicionar outro tipo de informação ao grafo, nomeadamente a direção em que o fluxo se deu. São opções que podem ser tomadas no futuro, embora no meu ponto de vista, a complexidade que é adicionada ao grafo com a substituição das atuais linhas, não se justifique.

Muitas opções podem ser estudadas referentes ao desenho dos grafos dado o alto nível de personalização que os seus elementos podem ser sujeitos recorrendo a SVG, CSS e à própria API da biblioteca ou através da criação de novas formas de visualização, dependendo das fontes de dados que futuramente o sistema Pulso/Discovery poderá ter acesso. É sempre necessário ponderar a adição de determinadas propriedades de informação ao grafo, enriquecendo-o sem que com isso se adicione demasiada complexidade visual de modo a influenciar a atenção do analista ou focar a atenção deste para propriedades irrelevantes.

8.4 Dificuldades enfrentadas

O desenho de grafos em larga escala, quando procura representar propriedades relativamente a centenas ou mesmo milhares de ativos de rede e as suas relações de vizinhança, é não só um motivante desafio como também tráz muitas restrições em termos de com-

[1] `http://bost.ocks.org/mike/fisheye/`
[2] `http://www.w3.org/TR/SVG/`

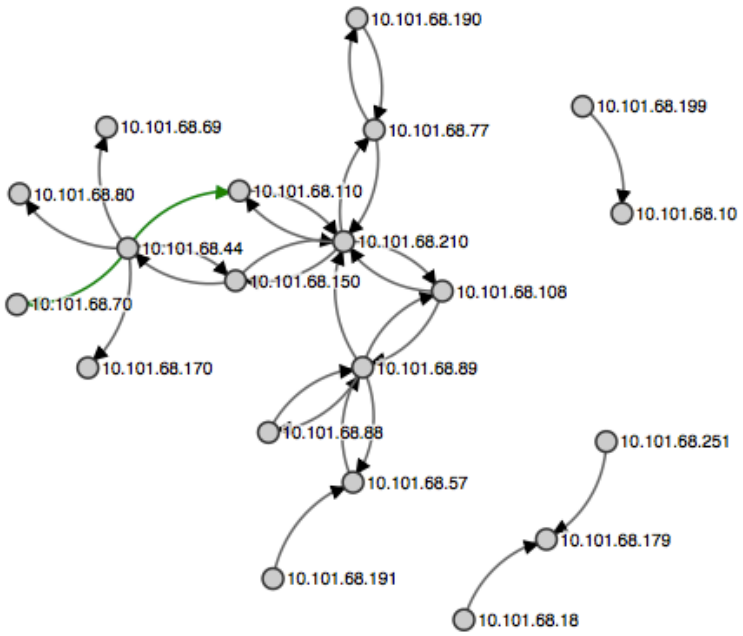

Figura 8.3: Exemplo de personalização do grafo de rede recorrendo a diferentes formas geométricas tais como setas para indicar a direção do fluxo de dados entre ativos.

plexidade computacional e cognitivas.

As dificuldades são acrescidas quando se recorre a um *front-end web* para gerar tal interface pessoa-máquina, conhecendo-se as limitações destes *front-ends*, nomeadamente a escassa gama de *frameworks* e ferramentas disponíveis para gerar visualizações e principalmente, as limitações para a criação de interações capazes de corresponder aos desafios de navegação por entre as propriedades de rede (ativos e fluxos de comunicação). A isto acrescenta-se limitações de eficiência de processos de desenho que *front-ends* desenvolvidos de raiz não apresentam.

No entanto, *front-ends web* têm vantagens que outras interfaces nativas não apresentam, para além da facilidade de integração futura do portal deste projeto com o da plataforma Pulso já existente.

Por outro lado, é necessário extrair propriedades úteis da grande quantidade de dados recolhidos pelas ferramentas de análise ativa e passiva de rede, por forma a identificar realmente o que é útil, disponibilizando visualizações e deduzindo eventos relevantes, permitindo posteriormente a identificação dos problemas quando estes surgem. Os dados capturados são imensos e todos os fluxos são relevantes refletindo-se em vários problemas de complexidade computacional para a plataforma. Grande parte dos dados não denunciam qualquer tipo de problemas relevantes, mas na grande maioria dos casos, apenas o analista tem a capacidade os identificar.

A falta de dados completos e consistentes capazes de gerar informação e visualizações úteis foi um problema. Navegar por grandes quantidades de dados é crucial para encon-

trar informação útil. Uma vez que o processo de recolha é tema de um outro projeto desenvolvido em paralelo com este, tais dados não se encontraram disponíveis para testar, dificultando em parte o desenho e a implementação deste. Assim, com a falta de dados concisos que ajudariam a compreender melhor a realidade do problema foi necessário ir adaptando técnicas e métodos à medida que estes ficavam disponíveis. Para lidar com este problema foi necessário recorrer a dados abstratos/previsíveis aquando do seu início, uma vez que a integração dos dois projetos Pulso/Discovery (sondas e servidor) foi um processo gradual e realizado mais tarde.

Diferentes ferramentas (autónomas) que analisam a rede, produzem resultados diferentes e muitas vezes com dados e graus de certeza contraditórios, capturados em diferentes intervalos temporais. Como o modelo de dados é alimentado por diferentes ferramentas independentes, foi necessário resolver problemas de cobertura de informação, e homogeneizar dados ambíguos ou a falta destes, contornando o problema omitindo ou deduzindo informação considerando o alto níveis de mudança a que a rede empresarial está sujeita.

Os dados por vezes incompletos ou com um grau de certeza baixo podem induzir problemas inexistentes, o que por pode tornar-se um problema dada a responsabilidade de fornecer informações corretas e consistentes.

Os testes gerais ao sistema foram realizados mais tarde do que previsto e muitas funcionalidades e melhorias descritas anteriormente, acabaram por não ser realizadas. A dependência que este projeto tem com o módulo responsável pela captura de tráfego de rede criou algumas limitações ao longo do seu desenvolvimento.

Uma vez que não foi possível, por parte das sondas Pulso/Discovery, analisar passivamente a rede em gamas de endereços de algumas aplicações (nomeadamente as sobre monitorização Pulso), não foram apresentados quaisquer evidências de comunicação entre estes ativos ou respetivos subsistemas e *clusters*. Em relação à análise de tráfego ativa da rede, dada a incompletude dos dados fornecidos pelas sondas, principalmente com escassez de propriedades referentes as ativos, os grafos de rede apresentados na secção 7.2.4 não foram os desejados, pois não há distinção entre tipos de ativos nem informação referente *hostnames* nos grafos, o que mudaria por completo a disposição e desenho destes. Escassez de dados e problemas estes, que fogem ao âmbito deste projeto.

Contudo, as dificuldades descritas anteriormente, que existem em qualquer projeto, foram de uma forma ou de outra contornadas contribuindo para um resultado final bastante positivo.

8.5 Conclusões

É importante referir que se cumpriu com o plano construído inicialmente e que se correspondeu com os objetivos propostos (secção 1.2) para este projeto.

Não houve qualquer desvio negativo no que diz respeito ao plano de trabalhos inicialmente definido, pode dizer-se que houve sim um desvio positivo, pois procurou-se utilizar tecnologias e métodos que não estavam previstos no seu início. O armazenamento de dados através de uma base de dados orientada a grafos, em detrimento de uma base de dados MySQL, foi um dos aspetos mais relevantes que não estava previsto, assim como o *cluster* de alta disponibilidade.

Além destas, muitas opções de arquitetura da plataforma Pulso/Discovery, tecnologias e bibliotecas foram estudadas e sendo propostas ao longo do desenvolvimento do projeto, o que se traduziu num investimento em termos de tempo na investigação, estudo, e principalmente na aprendizagem das diferentes API's e ferramentas. Este tempo extra, em certa parte previsto, não se refletiu em deslizes em termos de planeamento.

Muitas das opções arquiteturais do projeto foram tomadas por mim com sentido de responsabilidade de que no fim iria apresentar uma solução completamente funcional e de acordo com os objetivos propostos. É de realçar que estas opções se revelaram boas apostas, apresentando resultados bastante positivos e que proporcionam uma implementação bem estruturada, organizada e muito bem preparada para desenvolvimentos futuros.

Foi um projeto ambicioso, complexo e abrangente, onde existiu sempre melhorias e funcionalidades interessantes a acrescentar, algo que procurei fazer sempre que possível tendo em consideração o tempo disponibilizado e os objetivos propostos e centrais do projeto.

Apêndice A

Resultados dos testes apresentados

A.1 *Logs* do módulo DiscoveryServer

2012.09.24 11:13:48 DiscoveryServer Neo4j Discovery Server
2012.09.24 11:13:48 DiscoveryServer Reading property file
2012.09.24 11:13:48 DiscoveryServer Neo4j database interface set
2012.09.24 11:13:48 DiscoveryServer Connecting first instance to zookeeper: conf/ha1-neo4j.properties
2012.09.24 11:13:48 ZooKeeper Client environment:zookeeper.version=3.3.2-1031432, built on 11/05/2010 05:32 GMT
2012.09.24 11:13:48 ZooKeeper Client environment:host.name=discovery.tmn.pt
2012.09.24 11:13:48 ZooKeeper Client environment:java.version=1.6.0-29
2012.09.24 11:13:48 ZooKeeper Client environment:java.vendor=Sun Microsystems Inc.
2012.09.24 11:13:48 ZooKeeper Client environment:java.home=/xesta37/software/jdk1.6.0-29/jre
2012.09.24 11:13:48 ZooKeeper Client environment:java.class.path= lib/mysql-connector-java-5.1.20-bin.jar: lib/asm-3.1.jar: lib/json-lib-2-2.4-jdk15.jar: lib/neo4j-kernel-1.7.M02.jar: lib/geronimo-jta-1.1-spec-1.1.1.jar: lib/jsr311-api-1.1.1.jar: lib/neo4j-lucene-index-1.7.M02.jar: lib/jackson-core-asl-1.9.2.jar: lib/log4j-1.2.16.jar: lib/neo4j-management-1.7.M02.jar: lib/jackson-jaxrs-1.9.2.jar: lib/lucene-core-3.5.0.jar: lib/neo4j-shell-1.7.M02.jar: lib/jackson-mapper-asl-1.9.2.jar: lib/neo4j-backup-1.7.M02.jar: lib/neo4j-udc-1.7.M02.jar: lib/jackson-xc-1.9.2.jar: lib/neo4j-com-1.7.M02.jar: lib/org.apache.servicemix.bundles.jline-0.9.94-1.jar: lib/jersey-client-1.11.jar: lib/neo4j-cypher-1.7.M02.jar: lib/org.apache.servicemix.bundles.netty-3.2.5.Final-1.jar: lib/jersey-core-1.11.jar: lib/neo4j-graph-algo-1.7.M02.jar: lib/scala-library-2.9.0-1.jar: lib/jersey-json-1.11.jar: lib/neo4j-graph-matching-1.7.M02.jar: lib/server-api-1.7.M02.jar: lib/jersey-server-1.11.jar: lib/neo4j-ha-1.7.M02.jar: lib/slf4j-api-1.6.1.jar: lib/jersey-servlet-1.11.jar: lib/neo4j-jmx-1.7.M02.jar: lib/zookeeper-3.3.2.jar: lib/jettison-1.1.jar: lib/neo4j-kernel-1.7.M02-tests.jar: build/jar/DiscoveryServer.jar
2012.09.24 11:13:48 ZooKeeper Client environment:java.library.path=/xesta37/software/jdk1.6.0-29/jre/lib/amd64/server: /xesta37/software/jdk1.6.0-29/jre/lib/amd64: /xesta37/software/jdk1.6.0-29/jre/../lib/amd64:/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
2012.09.24 11:13:48 ZooKeeper Client environment:java.io.tmpdir=/tmp
2012.09.24 11:13:48 ZooKeeper Client environment:java.compiler=¡NA¿
2012.09.24 11:13:48 ZooKeeper Client environment:os.name=Linux
2012.09.24 11:13:48 ZooKeeper Client environment:os.arch=amd64
2012.09.24 11:13:48 ZooKeeper Client environment:os.version=2.6.18-128.el5
2012.09.24 11:13:48 ZooKeeper Client environment:user.name=root
2012.09.24 11:13:48 ZooKeeper Client environment:user.home=/root
2012.09.24 11:13:48 ZooKeeper Client environment:user.dir=/xesta37/DiscoveryServer
2012.09.24 11:13:48 ZooKeeper Initiating client connection, connectString=localhost:2181,localhost: 2182,localhost:2183 sessionTimeout=5000
2012.09.24 11:13:48 ZooKeeper watcher=org.neo4j.kernel.ha.zookeeper.ZooClient WatcherImpl@21ec6696
2012.09.24 11:13:48 ClientCnxn Opening socket connection to server localhost/127.0.0.1:2182
2012.09.24 11:13:48 ClientCnxn Socket connection established to localhost/127.0.0.1:2182, initiating session
2012.09.24 11:13:48 ZooKeeper Initiating client connection, connectString=localhost:2181,localhost: 2182,localhost:2183 sessionTimeout=5000
2012.09.24 11:13:48 ZooKeeper watcher=org.neo4j.kernel.ha.zookeeper.ZooKeeperClusterClient WatcherImpl@6cb8
2012.09.24 11:13:48 ClientCnxn Opening socket connection to server localhost/127.0.0.1:2183
2012.09.24 11:13:48 ClientCnxn Socket connection established to localhost/127.0.0.1:2183, initiating session
2012.09.24 11:13:48 ClientCnxn Session establishment complete on server localhost/127.0.0.1:2183, sessionid = 0x339f7c449ad0001, negotiated timeout = 6000
2012.09.24 11:13:48 ClientCnxn Session establishment complete on server localhost/127.0.0.1:2182, sessionid = 0x239f7c449ad0001, negotiated timeout = 6000
2012.09.24 11:13:48 ZooKeeper Session: 0x239f7c449ad0001 closed
2012.09.24 11:13:48 ClientCnxn EventThread shut down
2012.09.24 11:13:49 ZooKeeper Initiating client connection, connectString=localhost:2181,localhost: 2182,localhost:2183 sessionTimeout=5000
2012.09.24 11:13:49 ZooKeeper watcher=org.neo4j.kernel.ha.zookeeper.ZooClient WatcherImpl@77546dbc
2012.09.24 11:13:49 ClientCnxn Opening socket connection to server localhost/127.0.0.1:2181
2012.09.24 11:13:49 ClientCnxn Socket connection established to localhost/127.0.0.1:2181, initiating session
2012.09.24 11:13:49 ClientCnxn Session establishment complete on server localhost/127.0.0.1:2181, sessionid = 0x139f7c449ad0000, negotiated timeout = 6000
2012.09.24 11:13:50 DiscoveryServer Neo4j database connection done.
2012.09.24 11:13:50 DiscoveryServer Rails structure created sucessfully.
2012.09.24 11:13:54 DiscoveryServer Data added to db: 246 nodes, 237 relationships, 771 properties. (9 apps, 55 subsystems, 87 clusters, 95 hosts) (3337ms) from /10.101.0.15 (PulsoDB).
2012.09.24 11:13:55 DiscoveryServer Data added to db: 26 nodes, 16 relationships, 72 properties. (10 apps, 16 hosts) (459ms) from /10.101.0.15 (SOXmeta).
2012.09.24 11:13:55 DiscoveryServer Passive thread is listening: At port 6789
2012.09.24 11:13:55 DiscoveryServer Active thread is listening: At port 6666
2012.09.24 11:14:43 DiscoveryServer Data added to db: 223 nodes, 4430 relationships, 45433 properties. (17984ms) from /10.101.0.15 (Nmap).
2012.09.24 11:14:44 DiscoveryServer Data added to db: 32 nodes, 246 relationships, 2620 properties. (235ms) from /10.101.0.15 (Nmap).
2012.09.24 11:14:45 DiscoveryServer Data added to db: 0 nodes, 138 relationships, 1401 properties. (145ms) from /10.101.0.15 (Nmap).
2012.09.24 11:14:51 DiscoveryServer Data added to db: 81 nodes, 1015 relationships, 10558 properties. (1693ms) from /10.101.0.15 (Nmap).
2012.09.24 11:15:23 DiscoveryServer Data added to db: 53 nodes, 1632 relationships, 16585 properties. (6270ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:15:40 DiscoveryServer Data added to db: 15 nodes, 2963 relationships, 29705 properties. (12411ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:15:50 DiscoveryServer Data added to db: 14 nodes, 1690 relationships, 16970 properties. (5670ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:16:08 DiscoveryServer Data added to db: 5 nodes, 3028 relationships, 30305 properties. (12899ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:16:23 DiscoveryServer Data added to db: 8 nodes, 2590 relationships, 25940 properties. (9240ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:16:33 DiscoveryServer Data added to db: 14 nodes, 1683 relationships, 16900 properties. (5820ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:16:46 DiscoveryServer Data added to db: 10 nodes, 2191 relationships, 21960 properties. (9027ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:16:59 DiscoveryServer Data added to db: 4 nodes, 2177 relationships, 21790 properties. (8877ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:17:16 DiscoveryServer Data added to db: 9 nodes, 3402 relationships, 34065 properties. (11852ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:17:23 DiscoveryServer Data added to db: 8 nodes, 1721 relationships, 17250 properties. (3612ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:17:30 DiscoveryServer Data added to db: 3 nodes, 1597 relationships, 15985 properties. (2930ms) from /10.101.0.15 (IPAudit+P0F).

2012.09.24 11:17:39 DiscoveryServer Data added to db: 4 nodes, 1939 relationships, 19410 properties. (4634ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:17:43 DiscoveryServer Data added to db: 4 nodes, 1001 relationships, 10030 properties. (679ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:17:52 DiscoveryServer Data added to db: 2 nodes, 2047 relationships, 20480 properties. (4876ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:18:02 DiscoveryServer Data added to db: 2 nodes, 2270 relationships, 22710 properties. (5329ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:18:13 DiscoveryServer Data added to db: 2 nodes, 2502 relationships, 25030 properties. (7312ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:18:32 DiscoveryServer Data added to db: 5 nodes, 4542 relationships, 45445 properties. (12953ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:18:41 DiscoveryServer Data added to db: 5 nodes, 2341 relationships, 23435 properties. (5505ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:18:48 DiscoveryServer Data added to db: 0 nodes, 1574 relationships, 15740 properties. (2493ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:18:58 DiscoveryServer Data added to db: 4 nodes, 2491 relationships, 24930 properties. (6006ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:19:12 DiscoveryServer Data added to db: 1 nodes, 2740 relationships, 27405 properties. (9787ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:19:26 DiscoveryServer Data added to db: 2 nodes, 3134 relationships, 31350 properties. (9890ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:19:36 DiscoveryServer Data added to db: 26 nodes, 1978 relationships, 19910 properties. (5984ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:19:47 DiscoveryServer Data added to db: 3 nodes, 1827 relationships, 18285 properties. (7110ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:20:02 DiscoveryServer Data added to db: 8 nodes, 3276 relationships, 32800 properties. (9341ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:20:14 DiscoveryServer Data added to db: 19 nodes, 2750 relationships, 27595 properties. (8080ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:20:27 DiscoveryServer Data added to db: 5 nodes, 3036 relationships, 30385 properties. (8705ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:20:40 DiscoveryServer Data added to db: 5 nodes, 3260 relationships, 32625 properties. (8917ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:20:47 DiscoveryServer Data added to db: 1 nodes, 1618 relationships, 16185 properties. (3292ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:20:57 DiscoveryServer Data added to db: 9 nodes, 2223 relationships, 22275 properties. (5811ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:21:03 DiscoveryServer Data added to db: 1 nodes, 1667 relationships, 16675 properties. (2650ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:21:15 DiscoveryServer Data added to db: 7 nodes, 2341 relationships, 23445 properties. (7067ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:21:25 DiscoveryServer Data added to db: 3 nodes, 2081 relationships, 20825 properties. (6252ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:21:39 DiscoveryServer Data added to db: 3 nodes, 3228 relationships, 32295 properties. (8547ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:21:55 DiscoveryServer Data added to db: 3 nodes, 3422 relationships, 34235 properties. (9150ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:22:01 DiscoveryServer Data added to db: 8 nodes, 1450 relationships, 14540 properties. (2564ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:22:10 DiscoveryServer Data added to db: 1 nodes, 2097 relationships, 20975 properties. (5154ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:22:21 DiscoveryServer Data added to db: 3 nodes, 2643 relationships, 26445 properties. (6915ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:22:33 DiscoveryServer Data added to db: 6 nodes, 2190 relationships, 21930 properties. (7371ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:22:43 DiscoveryServer Data added to db: 1 nodes, 2258 relationships, 22585 properties. (6097ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:22:51 DiscoveryServer Data added to db: 2 nodes, 1997 relationships, 19980 properties. (4808ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:23:00 DiscoveryServer Data added to db: 1 nodes, 2033 relationships, 20335 properties. (5022ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:23:17 DiscoveryServer Data added to db: 1 nodes, 1992 relationships, 19925 properties. (5145ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:23:31 DiscoveryServer Data added to db: 0 nodes, 1753 relationships, 17530 properties. (3996ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:23:53 DiscoveryServer Data added to db: 3 nodes, 2177 relationships, 21785 properties. (5495ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:24:06 DiscoveryServer Data added to db: 3 nodes, 2154 relationships, 21555 properties. (5583ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:24:20 DiscoveryServer Data added to db: 1 nodes, 1786 relationships, 17865 properties. (3861ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:24:37 DiscoveryServer Data added to db: 1 nodes, 1984 relationships, 19845 properties. (4384ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:24:53 DiscoveryServer Data added to db: 0 nodes, 2332 relationships, 23320 properties. (5924ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:25:09 DiscoveryServer Data added to db: 1 nodes, 3036 relationships, 30365 properties. (9023ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:25:21 DiscoveryServer Data added to db: 1 nodes, 1690 relationships, 16905 properties. (5164ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:25:32 DiscoveryServer Data added to db: 2 nodes, 1529 relationships, 15300 properties. (2721ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:25:50 DiscoveryServer Data added to db: 0 nodes, 2567 relationships, 25670 properties. (7336ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:26:03 DiscoveryServer Data added to db: 1 nodes, 2294 relationships, 22945 properties. (5722ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:26:15 DiscoveryServer Data added to db: 1 nodes, 1437 relationships, 14375 properties. (2273ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:26:29 DiscoveryServer Data added to db: 0 nodes, 1122 relationships, 11220 properties. (1186ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:26:46 DiscoveryServer Data added to db: 0 nodes, 1688 relationships, 16880 properties. (3898ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:27:04 DiscoveryServer Data added to db: 1 nodes, 2087 relationships, 20875 properties. (5169ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:27:13 DiscoveryServer Data added to db: 0 nodes, 1267 relationships, 12670 properties. (1666ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:27:27 DiscoveryServer Data added to db: 1 nodes, 1235 relationships, 12355 properties. (2181ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:27:42 DiscoveryServer Data added to db: 2 nodes, 1485 relationships, 14860 properties. (2930ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:27:58 DiscoveryServer Data added to db: 2 nodes, 2151 relationships, 21520 properties. (6031ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:28:12 DiscoveryServer Data added to db: 1 nodes, 1674 relationships, 16745 properties. (3834ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:28:30 DiscoveryServer Data added to db: 2 nodes, 2000 relationships, 20010 properties. (5843ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:28:40 DiscoveryServer Data added to db: 1 nodes, 1180 relationships, 11805 properties. (1408ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:28:59 DiscoveryServer Data added to db: 1 nodes, 1536 relationships, 15365 properties. (2759ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:29:14 DiscoveryServer Data added to db: 0 nodes, 1100 relationships, 11000 properties. (958ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:29:33 DiscoveryServer Data added to db: 0 nodes, 1439 relationships, 14390 properties. (2419ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:29:48 DiscoveryServer Data added to db: 2 nodes, 1265 relationships, 12660 properties. (1739ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:30:08 DiscoveryServer Data added to db: 0 nodes, 1769 relationships, 17690 properties. (4174ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:30:22 DiscoveryServer Data added to db: 1 nodes, 1636 relationships, 16365 properties. (3410ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:30:38 DiscoveryServer Data added to db: 2 nodes, 1455 relationships, 14560 properties. (2028ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:30:53 DiscoveryServer Data added to db: 1 nodes, 1297 relationships, 12975 properties. (1796ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:31:10 DiscoveryServer Data added to db: 0 nodes, 998 relationships, 9980 properties. (463ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:31:37 DiscoveryServer Data added to db: 0 nodes, 4069 relationships, 40690 properties. (11847ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:31:52 DiscoveryServer Data added to db: 1 nodes, 3314 relationships, 33145 properties. (9387ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:31:57 DiscoveryServer Data added to db: 0 nodes, 1236 relationships, 12360 properties. (1519ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:32:09 DiscoveryServer Data added to db: 1 nodes, 1089 relationships, 10895 properties. (414ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:32:35 DiscoveryServer Data added to db: 1 nodes, 2873 relationships, 28735 properties. (8584ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:32:48 DiscoveryServer Data added to db: 3 nodes, 2584 relationships, 25855 properties. (5520ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:33:05 DiscoveryServer Data added to db: 0 nodes, 2509 relationships, 25090 properties. (7048ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:33:14 DiscoveryServer Data added to db: 0 nodes, 1247 relationships, 12470 properties. (1576ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:33:32 DiscoveryServer Data added to db: 0 nodes, 1693 relationships, 16930 properties. (4493ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:33:48 DiscoveryServer Data added to db: 1 nodes, 1637 relationships, 16375 properties. (2625ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:34:06 DiscoveryServer Data added to db: 1 nodes, 2210 relationships, 22105 properties. (7868ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:34:18 DiscoveryServer Data added to db: 1 nodes, 1637 relationships, 16375 properties. (3008ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:34:29 DiscoveryServer Data added to db: 0 nodes, 1163 relationships, 11630 properties. (1049ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:34:45 DiscoveryServer Data added to db: 5 nodes, 1458 relationships, 14605 properties. (2773ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:35:02 DiscoveryServer Data added to db: 1 nodes, 1830 relationships, 18305 properties. (4213ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:35:14 DiscoveryServer Data added to db: 2 nodes, 1253 relationships, 12540 properties. (1280ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:35:27 DiscoveryServer Data added to db: 0 nodes, 842 relationships, 8420 properties. (230ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:35:48 DiscoveryServer Data added to db: 1 nodes, 1178 relationships, 11785 properties. (2016ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:36:04 DiscoveryServer Data added to db: 0 nodes, 1513 relationships, 15130 properties. (3527ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:36:18 DiscoveryServer Data added to db: 2 nodes, 957 relationships, 9580 properties. (777ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:36:32 DiscoveryServer Data added to db: 1 nodes, 1138 relationships, 11385 properties. (1638ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:36:50 DiscoveryServer Data added to db: 1 nodes, 1464 relationships, 14645 properties. (2574ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:37:11 DiscoveryServer Data added to db: 1 nodes, 1331 relationships, 13315 properties. (4831ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:37:46 DiscoveryServer Data added to db: 0 nodes, 1376 relationships, 13760 properties. (4376ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:37:54 DiscoveryServer Data added to db: 1 nodes, 1840 relationships, 18405 properties. (3975ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:38:06 DiscoveryServer Data added to db: 1 nodes, 2659 relationships, 26595 properties. (7909ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:38:11 DiscoveryServer Data added to db: 1 nodes, 1322 relationships, 13225 properties. (1399ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:38:24 DiscoveryServer Data added to db: 1 nodes, 1837 relationships, 18375 properties. (3871ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:38:39 DiscoveryServer Data added to db: 0 nodes, 1067 relationships, 10670 properties. (1000ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:38:54 DiscoveryServer Data added to db: 1 nodes, 1260 relationships, 12605 properties. (1660ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:39:08 DiscoveryServer Data added to db: 2 nodes, 1072 relationships, 10730 properties. (953ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:39:26 DiscoveryServer Data added to db: 2 nodes, 1409 relationships, 14100 properties. (2887ms) from /10.101.0.15 (IPAudit+P0F).

2012.09.24 11:39:46 DiscoveryServer Data added to db: 0 nodes, 1356 relationships, 13560 properties. (2736ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:39:59 DiscoveryServer Data added to db: 0 nodes, 1088 relationships, 10880 properties. (1454ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:40:16 DiscoveryServer Data added to db: 0 nodes, 1330 relationships, 13300 properties. (4650ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:40:30 DiscoveryServer Data added to db: 3 nodes, 1348 relationships, 13495 properties. (4000ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:40:49 DiscoveryServer Data added to db: 3 nodes, 1777 relationships, 17785 properties. (4800ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:41:02 DiscoveryServer Data added to db: 1 nodes, 1554 relationships, 15545 properties. (3386ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:41:18 DiscoveryServer Data added to db: 2 nodes, 1581 relationships, 15820 properties. (4020ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:41:31 DiscoveryServer Data added to db: 0 nodes, 1103 relationships, 11030 properties. (1504ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:41:53 DiscoveryServer Data added to db: 1 nodes, 1242 relationships, 12425 properties. (3025ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:42:11 DiscoveryServer Data added to db: 1 nodes, 1776 relationships, 17765 properties. (3935ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:42:26 DiscoveryServer Data added to db: 2 nodes, 1433 relationships, 14340 properties. (2713ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:42:40 DiscoveryServer Data added to db: 0 nodes, 919 relationships, 9190 properties. (546ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:42:56 DiscoveryServer Data added to db: 3 nodes, 968 relationships, 9695 properties. (558ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:43:13 DiscoveryServer Data added to db: 2 nodes, 973 relationships, 9740 properties. (454ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:43:32 DiscoveryServer Data added to db: 0 nodes, 1388 relationships, 13880 properties. (3968ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:43:46 DiscoveryServer Data added to db: 1 nodes, 1059 relationships, 10595 properties. (1177ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:44:00 DiscoveryServer Data added to db: 2 nodes, 1043 relationships, 10440 properties. (1212ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:44:15 DiscoveryServer Data added to db: 1 nodes, 1336 relationships, 13365 properties. (2234ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:44:30 DiscoveryServer Data added to db: 1 nodes, 1431 relationships, 14315 properties. (3120ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:44:45 DiscoveryServer Data added to db: 1 nodes, 1241 relationships, 12415 properties. (1715ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:45:02 DiscoveryServer Data added to db: 1 nodes, 1703 relationships, 17035 properties. (3592ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:45:17 DiscoveryServer Data added to db: 0 nodes, 1925 relationships, 19250 properties. (5186ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:45:30 DiscoveryServer Data added to db: 0 nodes, 1080 relationships, 10800 properties. (3177ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:45:46 DiscoveryServer Data added to db: 4 nodes, 1374 relationships, 13760 properties. (2186ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:46:01 DiscoveryServer Data added to db: 0 nodes, 1459 relationships, 14590 properties. (2319ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:46:11 DiscoveryServer Data added to db: 1 nodes, 953 relationships, 9535 properties. (440ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:46:27 DiscoveryServer Data added to db: 0 nodes, 1514 relationships, 15140 properties. (3279ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:46:42 DiscoveryServer Data added to db: 1 nodes, 1265 relationships, 12655 properties. (2208ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:46:58 DiscoveryServer Data added to db: 1 nodes, 1284 relationships, 12845 properties. (2626ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:47:12 DiscoveryServer Data added to db: 0 nodes, 1346 relationships, 13460 properties. (2304ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:47:27 DiscoveryServer Data added to db: 2 nodes, 1180 relationships, 11810 properties. (1866ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:47:41 DiscoveryServer Data added to db: 1 nodes, 1185 relationships, 11855 properties. (1385ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:47:57 DiscoveryServer Data added to db: 1 nodes, 1095 relationships, 10955 properties. (2105ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:48:08 DiscoveryServer Data added to db: 0 nodes, 900 relationships, 9000 properties. (395ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:48:23 DiscoveryServer Data added to db: 1 nodes, 1086 relationships, 10865 properties. (1238ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:48:41 DiscoveryServer Data added to db: 0 nodes, 1708 relationships, 17080 properties. (2973ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:48:57 DiscoveryServer Data added to db: 1 nodes, 2245 relationships, 22455 properties. (5337ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:49:08 DiscoveryServer Data added to db: 0 nodes, 1584 relationships, 15840 properties. (2470ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:49:23 DiscoveryServer Data added to db: 0 nodes, 1442 relationships, 14420 properties. (1867ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:49:38 DiscoveryServer Data added to db: 0 nodes, 1299 relationships, 12990 properties. (2054ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:49:52 DiscoveryServer Data added to db: 1 nodes, 1423 relationships, 14235 properties. (3396ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:50:08 DiscoveryServer Data added to db: 1 nodes, 1476 relationships, 14765 properties. (3505ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:50:22 DiscoveryServer Data added to db: 2 nodes, 1393 relationships, 13940 properties. (2268ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:50:37 DiscoveryServer Data added to db: 0 nodes, 1182 relationships, 11820 properties. (1209ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:50:59 DiscoveryServer Data added to db: 0 nodes, 2321 relationships, 23210 properties. (6079ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:51:13 DiscoveryServer Data added to db: 0 nodes, 2496 relationships, 24960 properties. (6331ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:51:27 DiscoveryServer Data added to db: 1 nodes, 1339 relationships, 13395 properties. (3669ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:51:44 DiscoveryServer Data added to db: 2 nodes, 1051 relationships, 10520 properties. (1067ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:52:01 DiscoveryServer Data added to db: 1 nodes, 1621 relationships, 16215 properties. (3910ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:52:17 DiscoveryServer Data added to db: 1 nodes, 1114 relationships, 11145 properties. (607ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:52:33 DiscoveryServer Data added to db: 4 nodes, 950 relationships, 9520 properties. (440ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:52:53 DiscoveryServer Data added to db: 0 nodes, 1143 relationships, 11430 properties. (765ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:53:20 DiscoveryServer Data added to db: 1 nodes, 2345 relationships, 23455 properties. (5307ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:53:43 DiscoveryServer Data added to db: 0 nodes, 1362 relationships, 13620 properties. (1329ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:54:02 DiscoveryServer Data added to db: 3 nodes, 1817 relationships, 18185 properties. (3800ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:54:21 DiscoveryServer Data added to db: 0 nodes, 1043 relationships, 10430 properties. (800ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:54:48 DiscoveryServer Data added to db: 0 nodes, 1281 relationships, 12810 properties. (2697ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:55:06 DiscoveryServer Data added to db: 1 nodes, 1348 relationships, 13485 properties. (4100ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:55:25 DiscoveryServer Data added to db: 2 nodes, 1770 relationships, 17710 properties. (4765ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:55:42 DiscoveryServer Data added to db: 0 nodes, 1557 relationships, 15570 properties. (3390ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:56:05 DiscoveryServer Data added to db: 0 nodes, 1355 relationships, 13550 properties. (1469ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:56:11 DiscoveryServer Data added to db: 1 nodes, 1442 relationships, 14425 properties. (1868ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:56:16 DiscoveryServer Data added to db: 1 nodes, 1223 relationships, 12235 properties. (1261ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:56:31 DiscoveryServer Data added to db: 4 nodes, 3270 relationships, 32720 properties. (7615ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:56:38 DiscoveryServer Data added to db: 0 nodes, 1489 relationships, 14890 properties. (2743ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:57:00 DiscoveryServer Data added to db: 1 nodes, 1585 relationships, 15855 properties. (2887ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:57:25 DiscoveryServer Data added to db: 3 nodes, 1333 relationships, 13345 properties. (1788ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:57:41 DiscoveryServer Data added to db: 0 nodes, 1441 relationships, 14410 properties. (1866ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:57:58 DiscoveryServer Data added to db: 0 nodes, 955 relationships, 9550 properties. (451ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:58:17 DiscoveryServer Data added to db: 2 nodes, 1345 relationships, 13460 properties. (1678ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:58:31 DiscoveryServer Data added to db: 1 nodes, 1123 relationships, 11235 properties. (745ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:58:32 DiscoveryServer Data added to db: 1 nodes, 1459 relationships, 14595 properties. (2320ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:58:52 DiscoveryServer Data added to db: 0 nodes, 1466 relationships, 14660 properties. (2324ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:59:10 DiscoveryServer Data added to db: 3 nodes, 1723 relationships, 17245 properties. (4042ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:59:31 DiscoveryServer Data added to db: 0 nodes, 1031 relationships, 10310 properties. (1113ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 11:59:44 DiscoveryServer Data added to db: 1 nodes, 2783 relationships, 27835 properties. (5544ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:00:08 DiscoveryServer Data added to db: 3 nodes, 1226 relationships, 12275 properties. (2483ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:00:22 DiscoveryServer Data added to db: 2 nodes, 2883 relationships, 28840 properties. (6544ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:00:33 DiscoveryServer Data added to db: 4 nodes, 2526 relationships, 25280 properties. (4997ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:00:41 DiscoveryServer Data added to db: 0 nodes, 1713 relationships, 17130 properties. (4000ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:00:59 DiscoveryServer Data added to db: 1 nodes, 1955 relationships, 19555 properties. (2451ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:01:17 DiscoveryServer Data added to db: 2 nodes, 945 relationships, 9460 properties. (678ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:01:36 DiscoveryServer Data added to db: 0 nodes, 2124 relationships, 21240 properties. (2742ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:01:49 DiscoveryServer Data added to db: 1 nodes, 1109 relationships, 11095 properties. (1524ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:02:09 DiscoveryServer Data added to db: 0 nodes, 2192 relationships, 21920 properties. (5422ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:02:26 DiscoveryServer Data added to db: 0 nodes, 1634 relationships, 16340 properties. (2295ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:02:35 DiscoveryServer Data added to db: 0 nodes, 1021 relationships, 10210 properties. (1111ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:02:51 DiscoveryServer Data added to db: 1 nodes, 1785 relationships, 17855 properties. (3940ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:03:08 DiscoveryServer Data added to db: 3 nodes, 1336 relationships, 13375 properties. (2103ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:03:20 DiscoveryServer Data added to db: 1 nodes, 1001 relationships, 10015 properties. (911ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:03:38 DiscoveryServer Data added to db: 0 nodes, 1585 relationships, 15850 properties. (2887ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:03:51 DiscoveryServer Data added to db: 2 nodes, 1331 relationships, 13320 properties. (1893ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:04:10 DiscoveryServer Data added to db: 3 nodes, 2024 relationships, 20255 properties. (2880ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:04:25 DiscoveryServer Data added to db: 0 nodes, 3567 relationships, 35670 properties. (9240ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:04:44 DiscoveryServer Data added to db: 2 nodes, 1280 relationships, 12810 properties. (1754ms) from /10.101.0.15 (IPAudit+P0F).

2012.09.24 12:05:00 DiscoveryServer Data added to db: 0 nodes, 1245 relationships, 12450 properties. (1783ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:05:14 DiscoveryServer Data added to db: 0 nodes, 1459 relationships, 14590 properties. (2309ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:05:29 DiscoveryServer Data added to db: 1 nodes, 1920 relationships, 19205 properties. (5174ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:05:47 DiscoveryServer Data added to db: 0 nodes, 1085 relationships, 10850 properties. (3166ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:06:09 DiscoveryServer Data added to db: 0 nodes, 1384 relationships, 13840 properties. (2199ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:06:28 DiscoveryServer Data added to db: 0 nodes, 1419 relationships, 14190 properties. (2300ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:06:42 DiscoveryServer Data added to db: 0 nodes, 1053 relationships, 10530 properties. (245ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:06:57 DiscoveryServer Data added to db: 1 nodes, 1346 relationships, 13465 properties. (2304ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:07:23 DiscoveryServer Data added to db: 2 nodes, 1622 relationships, 16230 properties. (6273ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:07:45 DiscoveryServer Data added to db: 1 nodes, 2962 relationships, 29625 properties. (11451ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:08:05 DiscoveryServer Data added to db: 0 nodes, 1690 relationships, 16900 properties. (5620ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:08:25 DiscoveryServer Data added to db: 0 nodes, 3024 relationships, 30240 properties. (9899ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:08:43 DiscoveryServer Data added to db: 1 nodes, 2566 relationships, 25665 properties. (8240ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:08:59 DiscoveryServer Data added to db: 1 nodes, 1680 relationships, 16805 properties. (5754ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:09:18 DiscoveryServer Data added to db: 0 nodes, 2190 relationships, 21900 properties. (8377ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:09:32 DiscoveryServer Data added to db: 2 nodes, 2173 relationships, 21740 properties. (8170ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:09:49 DiscoveryServer Data added to db: 0 nodes, 3501 relationships, 35010 properties. (10852ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:10:12 DiscoveryServer Data added to db: 2 nodes, 1745 relationships, 17460 properties. (3610ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:10:29 DiscoveryServer Data added to db: 1 nodes, 1763 relationships, 17635 properties. (2920ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:10:44 DiscoveryServer Data added to db: 1 nodes, 1689 relationships, 16895 properties. (4640ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:11:04 DiscoveryServer Data added to db: 0 nodes, 1398 relationships, 13980 properties. (2271ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:11:26 DiscoveryServer Data added to db: 0 nodes, 1446 relationships, 14460 properties. (1878ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:11:47 DiscoveryServer Data added to db: 2 nodes, 1243 relationships, 12440 properties. (1260ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:12:05 DiscoveryServer Data added to db: 1 nodes, 3260 relationships, 32605 properties. (8630ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:12:23 DiscoveryServer Data added to db: 0 nodes, 1246 relationships, 12460 properties. (1478ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:12:40 DiscoveryServer Data added to db: 3 nodes, 1072 relationships, 10735 properties. (754ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:12:59 DiscoveryServer Data added to db: 1 nodes, 2027 relationships, 20275 properties. (4777ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:13:19 DiscoveryServer Data added to db: 1 nodes, 2070 relationships, 20705 properties. (5130ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:13:46 DiscoveryServer Data added to db: 0 nodes, 2402 relationships, 24020 properties. (4312ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:14:17 DiscoveryServer Data added to db: 0 nodes, 2542 relationships, 25420 properties. (4212ms) from /10.101.0.15 (IPAudit+P0F).
2012.09.24 12:14:38 DiscoveryServer Data added to db: 1 nodes, 1233 relationships, 12335 properties. (1265ms) from /10.101.0.15 (IPAudit+P0F).

A.1.1 Tabela derivada da informação dos *logs*

Tabela A.1: Tabela resultante da informação disponibilizada pelos *logs* do módulo Discovery Server.

Hora	Nós	Relações	Propriedades	Tempo de processamento (ms)	Fonte
2012.09.24 11:13:54	246	237	771	3337	PulsoDB
2012.09.24 11:13:55	26	16	72	459	SOXmeta
2012.09.24 11:14:43	223	4430	45433	17984	Nmap
2012.09.24 11:14:44	32	246	2620	235	Nmap
2012.09.24 11:14:45	0	138	1401	145	Nmap
2012.09.24 11:14:51	81	1015	10558	1693	Nmap
2012.09.24 11:15:23	53	1632	16585	6270	IPAudit+P0F
2012.09.24 11:15:40	15	2963	29705	12411	IPAudit+P0F
2012.09.24 11:15:50	14	1690	16970	5670	IPAudit+P0F
2012.09.24 11:16:08	5	3028	30305	12899	IPAudit+P0F
2012.09.24 11:16:23	8	2590	25940	9240	IPAudit+P0F
2012.09.24 11:16:33	14	1683	16900	5820	IPAudit+P0F
2012.09.24 11:16:46	10	2191	21960	9027	IPAudit+P0F
2012.09.24 11:16:59	4	2177	21790	8877	IPAudit+P0F
2012.09.24 11:17:16	9	3402	34065	11852	IPAudit+P0F
2012.09.24 11:17:23	8	1721	17250	3612	IPAudit+P0F
2012.09.24 11:17:30	3	1597	15985	2930	IPAudit+P0F
2012.09.24 11:17:39	4	1939	19410	4634	IPAudit+P0F
2012.09.24 11:17:43	4	1001	10030	679	IPAudit+P0F
2012.09.24 11:17:52	2	2047	20480	4876	IPAudit+P0F
2012.09.24 11:18:02	2	2270	22710	5329	IPAudit+P0F
2012.09.24 11:18:13	2	2502	25030	7312	IPAudit+P0F
2012.09.24 11:18:32	5	4542	45445	12953	IPAudit+P0F
2012.09.24 11:18:41	5	2341	23435	5505	IPAudit+P0F
2012.09.24 11:18:48	0	1574	15740	2493	IPAudit+P0F
2012.09.24 11:18:58	4	2491	24930	6006	IPAudit+P0F
2012.09.24 11:19:12	1	2740	27405	9787	IPAudit+P0F
2012.09.24 11:19:26	2	3134	31350	9890	IPAudit+P0F
2012.09.24 11:19:36	26	1978	19910	5984	IPAudit+P0F
2012.09.24 11:19:47	3	1827	18285	7110	IPAudit+P0F
2012.09.24 11:20:02	8	3276	32800	9341	IPAudit+P0F
2012.09.24 11:20:14	19	2750	27595	8080	IPAudit+P0F
2012.09.24 11:20:27	5	3036	30385	8705	IPAudit+P0F
2012.09.24 11:20:40	5	3260	32625	8917	IPAudit+P0F
2012.09.24 11:20:47	1	1618	16185	3292	IPAudit+P0F
2012.09.24 11:20:57	9	2223	22275	5811	IPAudit+P0F
2012.09.24 11:21:03	1	1667	16675	2650	IPAudit+P0F
2012.09.24 11:21:15	7	2341	23445	7067	IPAudit+P0F
2012.09.24 11:21:25	3	2081	20825	6252	IPAudit+P0F
2012.09.24 11:21:39	3	3228	32295	8547	IPAudit+P0F
2012.09.24 11:21:55	3	3422	34235	9150	IPAudit+P0F
2012.09.24 11:22:01	8	1450	14540	2564	IPAudit+P0F
2012.09.24 11:22:10	1	2097	20975	5154	IPAudit+P0F
2012.09.24 11:22:21	3	2643	26445	6915	IPAudit+P0F
2012.09.24 11:22:33	6	2190	21930	7371	IPAudit+P0F
2012.09.24 11:22:43	1	2258	22585	6097	IPAudit+P0F
2012.09.24 11:22:51	2	1997	19980	4808	IPAudit+P0F
2012.09.24 11:23:00	1	2033	20335	5022	IPAudit+P0F
2012.09.24 11:23:17	1	1992	19925	5145	IPAudit+P0F
2012.09.24 11:23:31	0	1753	17530	3996	IPAudit+P0F
2012.09.24 11:23:53	3	2177	21785	5495	IPAudit+P0F
2012.09.24 11:24:06	3	2154	21555	5583	IPAudit+P0F

Continua na proxima página

Tabela A.1 – continuação da página anterior

Hora	Nós	Relações	Propriedades	Tempo de processamento (ms)	Fonte
2012.09.24 11:24:20	1	1786	17865	3861	IPAudit+P0F
2012.09.24 11:24:37	1	1984	19845	4384	IPAudit+P0F
2012.09.24 11:24:53	0	2332	23320	5924	IPAudit+P0F
2012.09.24 11:25:09	1	3036	30365	9023	IPAudit+P0F
2012.09.24 11:25:21	1	1690	16905	5164	IPAudit+P0F
2012.09.24 11:25:32	2	1529	15300	2721	IPAudit+P0F
2012.09.24 11:25:50	0	2567	25670	7336	IPAudit+P0F
2012.09.24 11:26:03	1	2294	22945	5722	IPAudit+P0F
2012.09.24 11:26:15	1	1437	14375	2273	IPAudit+P0F
2012.09.24 11:26:29	0	1122	11220	1186	IPAudit+P0F
2012.09.24 11:26:46	0	1688	16880	3898	IPAudit+P0F
2012.09.24 11:27:04	1	2087	20875	5169	IPAudit+P0F
2012.09.24 11:27:13	0	1267	12670	1666	IPAudit+P0F
2012.09.24 11:27:27	1	1235	12355	2181	IPAudit+P0F
2012.09.24 11:27:42	2	1485	14860	2930	IPAudit+P0F
2012.09.24 11:27:58	2	2151	21520	6031	IPAudit+P0F
2012.09.24 11:28:12	1	1674	16745	3834	IPAudit+P0F
2012.09.24 11:28:30	2	2000	20010	5843	IPAudit+P0F
2012.09.24 11:28:40	1	1180	11805	1408	IPAudit+P0F
2012.09.24 11:28:59	1	1536	15365	2759	IPAudit+P0F
2012.09.24 11:29:14	0	1100	11000	958	IPAudit+P0F
2012.09.24 11:29:33	0	1439	14390	2419	IPAudit+P0F
2012.09.24 11:29:48	2	1265	12660	1739	IPAudit+P0F
2012.09.24 11:30:08	0	1769	17690	4174	IPAudit+P0F
2012.09.24 11:30:22	1	1636	16365	3410	IPAudit+P0F
2012.09.24 11:30:38	2	1455	14560	2028	IPAudit+P0F
2012.09.24 11:30:53	1	1297	12975	1796	IPAudit+P0F
2012.09.24 11:31:10	0	998	9980	463	IPAudit+P0F
2012.09.24 11:31:37	0	4069	40690	11847	IPAudit+P0F
2012.09.24 11:31:52	1	3314	33145	9387	IPAudit+P0F
2012.09.24 11:31:57	0	1236	12360	1519	IPAudit+P0F
2012.09.24 11:32:09	1	1089	10895	414	IPAudit+P0F
2012.09.24 11:32:35	1	2873	28735	8584	IPAudit+P0F
2012.09.24 11:32:48	3	2584	25855	5520	IPAudit+P0F
2012.09.24 11:33:05	0	2509	25090	7048	IPAudit+P0F
2012.09.24 11:33:14	0	1247	12470	1576	IPAudit+P0F
2012.09.24 11:33:32	0	1693	16930	4493	IPAudit+P0F
2012.09.24 11:33:48	1	1637	16375	2625	IPAudit+P0F
2012.09.24 11:34:06	1	2210	22105	7868	IPAudit+P0F
2012.09.24 11:34:18	1	1637	16375	3008	IPAudit+P0F
2012.09.24 11:34:29	0	1163	11630	1049	IPAudit+P0F
2012.09.24 11:34:45	5	1458	14605	2773	IPAudit+P0F
2012.09.24 11:35:02	1	1830	18305	4213	IPAudit+P0F
2012.09.24 11:35:14	2	1253	12540	1280	IPAudit+P0F
2012.09.24 11:35:27	0	842	8420	230	IPAudit+P0F
2012.09.24 11:35:48	1	1178	11785	2016	IPAudit+P0F
2012.09.24 11:36:04	0	1513	15130	3527	IPAudit+P0F
2012.09.24 11:36:18	2	957	9580	777	IPAudit+P0F
2012.09.24 11:36:32	1	1138	11385	1638	IPAudit+P0F
2012.09.24 11:36:50	1	1464	14645	2574	IPAudit+P0F
2012.09.24 11:37:11	1	1331	13315	4831	IPAudit+P0F
2012.09.24 11:37:46	0	1376	13760	4376	IPAudit+P0F
2012.09.24 11:37:54	1	1840	18405	3975	IPAudit+P0F
2012.09.24 11:38:06	1	2659	26595	7909	IPAudit+P0F
2012.09.24 11:38:11	1	1322	13225	1399	IPAudit+P0F
2012.09.24 11:38:24	1	1837	18375	3871	IPAudit+P0F
2012.09.24 11:38:39	0	1067	10670	1000	IPAudit+P0F
2012.09.24 11:38:54	1	1260	12605	1660	IPAudit+P0F
2012.09.24 11:39:08	2	1072	10730	953	IPAudit+P0F
2012.09.24 11:39:26	2	1409	14100	2887	IPAudit+P0F
2012.09.24 11:39:46	0	1356	13560	2736	IPAudit+P0F
2012.09.24 11:39:59	0	1088	10880	1454	IPAudit+P0F
2012.09.24 11:40:16	0	1330	13300	4650	IPAudit+P0F
2012.09.24 11:40:30	3	1348	13495	4000	IPAudit+P0F
2012.09.24 11:40:49	3	1777	17785	4800	IPAudit+P0F
2012.09.24 11:41:02	1	1554	15545	3386	IPAudit+P0F
2012.09.24 11:41:18	2	1581	15820	4020	IPAudit+P0F
2012.09.24 11:41:31	0	1103	11030	1504	IPAudit+P0F
2012.09.24 11:41:53	1	1242	12425	3025	IPAudit+P0F
2012.09.24 11:42:11	1	1776	17765	3935	IPAudit+P0F
2012.09.24 11:42:26	2	1433	14340	2713	IPAudit+P0F
2012.09.24 11:42:40	0	919	9190	546	IPAudit+P0F
2012.09.24 11:42:56	3	968	9695	558	IPAudit+P0F
2012.09.24 11:43:13	2	973	9740	454	IPAudit+P0F
2012.09.24 11:43:32	0	1388	13880	3968	IPAudit+P0F
2012.09.24 11:43:46	1	1059	10595	1177	IPAudit+P0F
2012.09.24 11:44:00	2	1043	10440	1212	IPAudit+P0F
2012.09.24 11:44:15	1	1336	13365	2234	IPAudit+P0F
2012.09.24 11:44:30	1	1431	14315	3120	IPAudit+P0F
2012.09.24 11:44:45	1	1241	12415	1715	IPAudit+P0F
2012.09.24 11:45:02	1	1703	17035	3592	IPAudit+P0F
2012.09.24 11:45:17	0	1925	19250	5186	IPAudit+P0F
2012.09.24 11:45:30	0	1080	10800	3177	IPAudit+P0F
2012.09.24 11:45:46	4	1374	13760	2186	IPAudit+P0F
2012.09.24 11:46:01	0	1459	14590	2319	IPAudit+P0F
2012.09.24 11:46:11	1	953	9535	440	IPAudit+P0F
2012.09.24 11:46:27	0	1514	15140	3279	IPAudit+P0F
2012.09.24 11:46:42	1	1265	12655	2208	IPAudit+P0F
2012.09.24 11:46:58	1	1284	12845	2626	IPAudit+P0F
2012.09.24 11:47:12	0	1346	13460	2304	IPAudit+P0F
2012.09.24 11:47:27	2	1180	11810	1866	IPAudit+P0F
2012.09.24 11:47:41	1	1185	11855	1385	IPAudit+P0F
2012.09.24 11:47:57	1	1095	10955	2105	IPAudit+P0F

Continua na proxima página

Tabela A.1 – continuação da página anterior

Hora	Nós	Relações	Propriedades	Tempo de processamento (ms)	Fonte
2012.09.24 11:48:08	0	900	9000	395	IPAudit+P0F
2012.09.24 11:48:23	1	1086	10865	1238	IPAudit+P0F
2012.09.24 11:48:41	0	1708	17080	2973	IPAudit+P0F
2012.09.24 11:48:57	1	2245	22455	5337	IPAudit+P0F
2012.09.24 11:49:08	0	1584	15840	2470	IPAudit+P0F
2012.09.24 11:49:23	0	1442	14420	1867	IPAudit+P0F
2012.09.24 11:49:38	0	1299	12990	2054	IPAudit+P0F
2012.09.24 11:49:52	1	1423	14235	3396	IPAudit+P0F
2012.09.24 11:50:08	1	1476	14765	3505	IPAudit+P0F
2012.09.24 11:50:22	2	1393	13940	2268	IPAudit+P0F
2012.09.24 11:50:37	0	1182	11820	1209	IPAudit+P0F
2012.09.24 11:50:59	0	2321	23210	6079	IPAudit+P0F
2012.09.24 11:51:13	0	2496	24960	6331	IPAudit+P0F
2012.09.24 11:51:27	1	1339	13395	3669	IPAudit+P0F
2012.09.24 11:51:44	2	1051	10520	1067	IPAudit+P0F
2012.09.24 11:52:01	1	1621	16215	3910	IPAudit+P0F
2012.09.24 11:52:17	1	1114	11145	607	IPAudit+P0F
2012.09.24 11:52:33	4	950	9520	440	IPAudit+P0F
2012.09.24 11:52:53	0	1143	11430	765	IPAudit+P0F
2012.09.24 11:53:20	1	2345	23455	5307	IPAudit+P0F
2012.09.24 11:53:43	0	1362	13620	1329	IPAudit+P0F
2012.09.24 11:54:02	3	1817	18185	3800	IPAudit+P0F
2012.09.24 11:54:21	0	1043	10430	800	IPAudit+P0F
2012.09.24 11:54:48	0	1281	12810	2697	IPAudit+P0F
2012.09.24 11:55:06	1	1348	13485	4100	IPAudit+P0F
2012.09.24 11:55:25	2	1770	17710	4765	IPAudit+P0F
2012.09.24 11:55:42	0	1557	15570	3390	IPAudit+P0F
2012.09.24 11:56:05	0	1355	13550	1469	IPAudit+P0F
2012.09.24 11:56:11	1	1442	14425	1868	IPAudit+P0F
2012.09.24 11:56:16	1	1223	12235	1261	IPAudit+P0F
2012.09.24 11:56:31	4	3270	32720	7615	IPAudit+P0F
2012.09.24 11:56:38	0	1489	14890	2743	IPAudit+P0F
2012.09.24 11:57:00	1	1585	15855	2887	IPAudit+P0F
2012.09.24 11:57:25	3	1333	13345	1788	IPAudit+P0F
2012.09.24 11:57:41	0	1441	14410	1866	IPAudit+P0F
2012.09.24 11:57:58	0	955	9550	451	IPAudit+P0F
2012.09.24 11:58:17	2	1345	13460	1678	IPAudit+P0F
2012.09.24 11:58:31	1	1123	11235	745	IPAudit+P0F
2012.09.24 11:58:32	1	1459	14595	2320	IPAudit+P0F
2012.09.24 11:58:52	0	1466	14660	2324	IPAudit+P0F
2012.09.24 11:59:10	3	1723	17245	4042	IPAudit+P0F
2012.09.24 11:59:31	0	1031	10310	1113	IPAudit+P0F
2012.09.24 11:59:44	1	2783	27835	5544	IPAudit+P0F
2012.09.24 12:00:08	3	1226	12275	2483	IPAudit+P0F
2012.09.24 12:00:22	2	2883	28840	6544	IPAudit+P0F
2012.09.24 12:00:33	4	2526	25280	4997	IPAudit+P0F
2012.09.24 12:00:41	0	1713	17130	4000	IPAudit+P0F
2012.09.24 12:00:59	1	1955	19555	2451	IPAudit+P0F
2012.09.24 12:01:17	2	945	9460	678	IPAudit+P0F
2012.09.24 12:01:36	0	2124	21240	2742	IPAudit+P0F
2012.09.24 12:01:49	1	1109	11095	1524	IPAudit+P0F
2012.09.24 12:02:09	0	2192	21920	5422	IPAudit+P0F
2012.09.24 12:02:26	0	1634	16340	2295	IPAudit+P0F
2012.09.24 12:02:35	0	1021	10210	1111	IPAudit+P0F
2012.09.24 12:02:51	1	1785	17855	3940	IPAudit+P0F
2012.09.24 12:03:08	3	1336	13375	2103	IPAudit+P0F
2012.09.24 12:03:20	1	1001	10015	911	IPAudit+P0F
2012.09.24 12:03:38	0	1585	15850	2887	IPAudit+P0F
2012.09.24 12:03:51	2	1331	13320	1893	IPAudit+P0F
2012.09.24 12:04:10	3	2024	20255	2880	IPAudit+P0F
2012.09.24 12:04:25	0	3567	35670	9240	IPAudit+P0F
2012.09.24 12:04:44	2	1280	12810	1754	IPAudit+P0F
2012.09.24 12:05:00	0	1245	12450	1783	IPAudit+P0F
2012.09.24 12:05:14	0	1459	14590	2309	IPAudit+P0F
2012.09.24 12:05:29	1	1920	19205	5174	IPAudit+P0F
2012.09.24 12:05:47	0	1085	10850	3166	IPAudit+P0F
2012.09.24 12:06:09	0	1384	13840	2199	IPAudit+P0F
2012.09.24 12:06:28	0	1419	14190	2300	IPAudit+P0F
2012.09.24 12:06:42	0	1053	10530	245	IPAudit+P0F
2012.09.24 12:06:57	1	1346	13465	2304	IPAudit+P0F
2012.09.24 12:07:23	2	1622	16230	6273	IPAudit+P0F
2012.09.24 12:07:45	1	2962	29625	11451	IPAudit+P0F
2012.09.24 12:08:05	0	1690	16900	5620	IPAudit+P0F
2012.09.24 12:08:25	0	3024	30240	9899	IPAudit+P0F
2012.09.24 12:08:43	1	2566	25665	8240	IPAudit+P0F
2012.09.24 12:08:59	1	1680	16805	5754	IPAudit+P0F
2012.09.24 12:09:18	0	2190	21900	8377	IPAudit+P0F
2012.09.24 12:09:32	2	2173	21740	8170	IPAudit+P0F
2012.09.24 12:09:49	0	3501	35010	10852	IPAudit+P0F
2012.09.24 12:10:12	2	1745	17460	3610	IPAudit+P0F
2012.09.24 12:10:29	1	1763	17635	2920	IPAudit+P0F
2012.09.24 12:10:44	1	1689	16895	4640	IPAudit+P0F
2012.09.24 12:11:04	0	1398	13980	2271	IPAudit+P0F
2012.09.24 12:11:26	0	1446	14460	1878	IPAudit+P0F
2012.09.24 12:11:47	2	1243	12440	1260	IPAudit+P0F
2012.09.24 12:12:05	1	3260	32605	8630	IPAudit+P0F
2012.09.24 12:12:23	0	1246	12460	1478	IPAudit+P0F
2012.09.24 12:12:40	3	1072	10735	754	IPAudit+P0F
2012.09.24 12:12:59	1	2027	20275	4777	IPAudit+P0F
2012.09.24 12:13:19	1	2070	20705	5130	IPAudit+P0F
2012.09.24 12:13:46	0	2402	24020	4312	IPAudit+P0F
2012.09.24 12:14:17	0	2542	25420	4212	IPAudit+P0F
2012.09.24 12:14:38	1	1233	12335	1265	IPAudit+P0F

A.2 Excerto dos *logs* do módulo DiscoveryPortal

```
== Booting WEBrick
== Rails 3.1.3 application starting in development on http://0.0.0.0:3000
== Call with -d to detach
== Ctrl-C to shutdown server
[2012-09-24 11:12:55] INFO WEBrick 1.3.1
[2012-09-24 11:12:55] INFO ruby 1.8.7 (2012-02-22) [java]
[2012-09-24 11:12:55] INFO WEBrick::HTTPServer#start: pid=13933 port=3000 starting Neo4j in HA mode, machine id: 2 at localhost:6002 db
/xesta37/DiscoveryZookeeper/Neo4jServer2.Rails/data/db

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:22:29 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: none
Method Time elapsed 0.163 seconds
Method + JSON Time elapsed 0.303 seconds
founded 46 non ranged hosts. 53 total (7 belong to apps)
founded 66 links (1005 flows total; maxlink: 175 flows)
Completed 200 OK in 347ms (Views: 71.0ms)

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:23:51 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: [10-10].[101-101].[0-255].[0-255]
Method Time elapsed 0.401 seconds
Method + JSON Time elapsed 0.467 seconds
founded 26 non ranged hosts. 64 total (0 belong to apps)
founded 79 links (1261 flows total; maxlink: 637 flows)
Completed 200 OK in 491ms (Views: 38.0ms)

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:24:28 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: 10.101.1.60
Method Time elapsed 0.057 seconds
Method + JSON Time elapsed 0.679 seconds
founded 52 non ranged hosts. 53 total (0 belong to apps)
founded 52 links (6274 flows total; maxlink: 1345 flows)
Completed 200 OK in 697ms (Views: 28.0ms)

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:38:20 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: none
Method Time elapsed 1.706 seconds
Method + JSON Time elapsed 4.661 seconds
founded 167 non ranged hosts. 175 total (8 belong to apps)
founded 245 links (63637 flows total; maxlink: 12291 flows)
Completed 200 OK in 4691ms (Views: 95.0ms)

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:42:09 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: [10-10].[101-101].[0-255].[0-255]
Method Time elapsed 1.203 seconds
Method + JSON Time elapsed 7.606 seconds
founded 107 non ranged hosts. 332 total (0 belong to apps)
founded 511 links (22029 flows total; maxlink: 9473 flows)
Completed 200 OK in 7667ms (Views: 102.0ms)

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:44:45 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: 10.101.1.60
Method Time elapsed 0.073 seconds
Method + JSON Time elapsed 3.703 seconds
founded 67 non ranged hosts. 68 total (0 belong to apps)
founded 67 links (67381 flows total; maxlink: 16137 flows)
Completed 200 OK in 3716ms (Views: 20.0ms)

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:53:44 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: none
Method Time elapsed 3.324 seconds
Method + JSON Time elapsed 8.389 seconds
founded 251 non ranged hosts. 259 total (8 belong to apps)
founded 352 links (115065 flows total; maxlink: 21701 flows)
Completed 200 OK in 8428ms (Views: 65.0ms)

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:57:58 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: [10-10].[101-101].[0-255].[0-255]
Method Time elapsed 1.924 seconds
Method + JSON Time elapsed 14.552 seconds
founded 147 non ranged hosts. 377 total (0 belong to apps)
founded 575 links (36206 flows total; maxlink: 14609 flows)
Completed 200 OK in 14616ms (Views: 105.0ms)

Started GET "/hosts.json"for 10.101.68.53 at Mon Sep 24 11:57:05 +0100 2012
Processing by HostsController#index as JSON
Range/IP searched: 10.101.1.60
Method Time elapsed 0.008 seconds
Method + JSON Time elapsed 6.785 seconds
founded 72 non ranged hosts. 73 total (0 belong to apps)
founded 72 links (104005 flows total; maxlink: 25213 flows)
Completed 200 OK in 6796ms (Views: 18.0ms)
```

Bibliografia

[1] Tiago F. R. Alegria José Alegria, José Carvalho, Tiago F. R. Carvalho, and Ricardo G. Ramalho. Uma experiência open source para ”tomar o pulso” e ”ter pulso” sobre a função sistemas e tecnologias de informação. 2005.

[2] J. Ignacio Alvarez-Hamelin, Luca Dall'Asta, and Alain Barrat Alessandro Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. Technical report, Université de Paris-Sud, 2006.

[3] Josh Barnes and Piet Hut. A hierarchical o(n log n) force calculation algorithm. *Journal of Computational Physics*, 1986.

[4] E.A Barnett. *Analytical Hypnotherapy: Principles and Practice*. CA: Westwood Publishing Company, 1989.

[5] Rick Cattell. Scalable SQL and NoSQL data stores. Technical report, 2011.

[6] Matthew Chalmers. A linear iteration time layout algorithm for visualising high–dimensional data. Technical report, UBILAB, Union Bank of Switzerland, 1996.

[7] Jeffrey B. Colombe and Gregory Stephens. Statistical profiling and visualization for detection of malicious insider attacks on computer networks. Technical report, 2004.

[8] Greg Conti. *Security Data Visualization: Graphical Techniques for Network Analysis*. 2007.

[9] João Dias Santa de Vasconcelos Duarte. Evaluating information systems- constructing a model processing framework. Master's thesis, Instituto Superior Técnico, 2009.

[10] Athanasios Douitsis and Dimitrios Kalogeras. Interactive network management visualization with svg and ajax. Technical report, National Technical University of Athens, 2006.

[11] P. Eades. A heuristic for graph drawning. Technical report, 1984.

[12] Denise Ferebee and Dipankar Dasgupta. Security visualization survey. Technical report, University of Memphis, 2008.

[13] Fabian Fischer, Florian Mansmann, Daniel A. Keim, Stephan Pietzko, and Marcel Waldvogel. Large-scale network monitoring for visual analysis of attacks. Technical report, University of Konstanz, 2008.

[14] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. Technical report, University of Illinois, 1991.

[15] M. Ghoniem, J. D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. Technical report, Ecole des Mines de Nantes, 2005.

[16] Nicklaus A. Giacobe, Vincent F. Mancuso, and Dev Minotra. Large scale network security visualization. Technical report, College of Information Sciences and Technology, 2010.

[17] Luc Girardin and Ubilab Dominique Brodbeck, UBS. A visual approach for monitoring logs. Technical report, UBS, Ubilab, 1998.

[18] Danny Holten and Jarke J. van Wijk. Force-directed edge bundling for graph visualization. Technical report, Eindhoven University of Technology, 2009.

[19] B. Huffaker, E. Nemeth, and k. claffy. A general-purpose network visualization tool. In *International Networking Conference (INET) '99*. The Internet Society, 1999.

[20] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. Technical report, 2010.

[21] Stephen F. Ingram. An interactive small world graph visualization. Technical report, University of British Columbia, 2005.

[22] Maxim Korotkov. Automatic layout of state diagrams. Technical report, 2008.

[23] David Llewellyn-Jones. Whole body interaction – position paper security visualisation and control using virtual environments. Technical report, School of Computing and Mathematical Sciences, 2007.

[24] Florian Mansmann, Lorenz Meier, and Daniel A. Keim. Visualization of host behavior for network security. Technical report, University of Konstanz, 2007.

[25] Florian Mansmann and Svetlana Vinnik. Interactive exploration of data traffic with hierarchical network maps. Technical report, University of Konstanz, 2006.

[26] Rui Martins. Descoberta automática das interdependências aplicacionais em grandes redes corporativas por analise passiva dos seus network flows. Master's thesis, Faculdade de Ciências da Universidade de Lisboa, 2011.

[27] Raffaell Marty. *Applied Security Visualization*. Addison-Wesley, 2008.

[28] Chris Muelder, Kwan-Liu Ma, and Tony Bartoletti. A visualization methodology for characterization of network scans. Technical report, University of California, 2005.

[29] Tamara Munzner. *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University, 2000.

[30] Andreas Noack. An energy model for visual graph clustering. Technical report, Brandenburg Technical University at Cottbus, 2003.

[31] Martin Roesch. Snort - lightweight intrusion detection for networks. Technical report, The USENIX Association, 1999.

[32] Tiago Simões. Investigação e desenvolvimento de um sistema automático de detecção, monitorização e análise da propagação de worms em redes empresarias. Master's thesis, Faculdade de Ciências da Universidade de Lisboa, 2011.

[33] Cynthia Smith. Cost-effective database scalability using database sharding. Technical report, Database Sharding White Paper, 2009.

[34] Christof Strauch. NoSQL databases. Technical report, Stuttgart Media University, 2011.

[35] Roberto Tamassia, Bernardo Palazzi, and Charalampos Papamanthou. Graph drawing for security visualization. Technical report, Brown University, Department of Computer Science, 2009.

[36] The Neo4j Team. *The Neo4j Manual 1.7*. Neo Technology, 2012.

[37] Jens Tölle and Oliver Niggemann. Supporting intrusion detection by graph clustering and graph drawing. Technical report, University of Bonn, 2000.

[38] Frank van Ham. Using multilevel call matrices in large software projects. Technical report, Technische Universiteit Eindhoven, 2003.

[39] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufman, 2004.

[40] Colin Ware and Robert Bobrow. Supporting visual queries on medium sized node-link diagrams. Technical report, 2005.